TRAINING FOR TRANSPUTER TECHNOLOGIES

A.W.P. Bakkers, R. Bruis, H.W. Roebbers, N.C. Schaller,
M.H. Schwirtz, J.P.E. Sunter, and K.C.J. Wijbrans[*]

Control Laboratory,Electrical Engineering Department
University of Twente, The Netherlands

P.H. Welch

Computing Laboratory
University of Kent,Canterbury, UK

1 Introduction

With the advent of the transputer in 1984, it has become feasible to create very powerful parallel processing systems at a moderate cost. The architecture of the transputer removes the need for special communication processors or shared memory buses, thus reducing the cost of cabling and components. Especially in embedded systems, the potential benefits of transputers are high. Because the component itself was designed for simplicity few additional components are needed. However, parallel processing requires the creation of parallel software. This is still perceived as a problem by most people and indeed it is a problem for conventional computer architectures. But unlike these, the transputer was designed specifically for handling parallelism. The transputer environment not only consists of the component, but also of the language Occam. This language and the transputer were designed together, resulting in efficient compilation of Occam code for the transputer[4]. Even more important is the solid theoretical foundation of the Occam language. Unlike most existing languages Occam has a mathematical strictness. It is based on the model of parallelism found in Communication Sequential Processes[3]. As a result, the mathematical correctness of Occam programs is provable by applying transformation rules. Thus, it can be concluded that all ingredients that are needed to make parallel processing feasible, are combined here: an efficient processor capable of handling parallelism and multi–processing, a programming language that directly addresses parallelism and a solid theoretical foundation.

Still, the reception of the transputer in industry was sceptical. Here, the opinion prevails that parallel processing is difficult. Therefore, we perceived a need for education in parallel processing for this specific target group. This target group has the following properties:

– Its members have an engineering background, for example in mechanical or electrical engineering or in computer science. The users function at least at a B.Sc. level, either by experience or by education.
– Its members have several years of experience in the use of *embedded computers*. Most of them are well familiar with existing microprocessor families, operating systems, programming in high–level languages and in assembler, and the use of interrupts and simple multi–tasking kernels.
– In general, its members are well familiar with (academic) developments in the computer world through engineering journals.
– Its members have a notion that parallel processing is difficult. This is partly because most publications on parallel processing concern huge dedicated systems with peculiar architectures (e.g., the connection machine, the BBN butterfly et cetera), partly because their own experience with multi–tasking operating systems (e.g., OS/9, VxWorks or VRTX) and interrupts confirms this opinion.

In this paper, we will discuss our experiences with the creation and presentation of the 'Training for Transputer Technology' courses on parallel processing. These courses are designed for industrial engineers. The material presented in the courses is selected on the basis of its practical usefulness and over 50% of the total course time is devoted to practical exercises. Section 2 will discuss the ancestry of these courses. In section 3, we will document the contents of the courses with an emphasis on their hands-on focus and the selection of exercises. Section 4 presents the reactions of participants of the course and their suggestions for improvement. Finally, some conclusions on the courses are given in section 5.

[*] At the time of the development of these courses detached at the Control Laboratory by Van Rietschoten & Houwens, Rotterdam, The Netherlands.

2 Course ancestry

September 1986 saw the first offering of the COMETT (COMmunities action programme for Education and Training for Technologies) course, "Basic Occam and Transputer Engineering" at the University of Kent at Canterbury, England. Since then, this course has been picked up by the Control Laboratory of the Electrical Engineering department of the University of Twente, and modified for the control engineering curriculum. After that, an industrial variant was created. This variant has been offered in the Netherlands, France, Belgium, Germany, the United States, South Africa, Greece, and Australia. A follow-up course, "Advanced Occam and Transputer Engineering", and an introductory workshop, "Design of a Parallel Stepper Motor Controller", have been added to the COMETT offerings. These three courses are included in a series of courses and technical workshops entitled `Training for Transputer Technologies', which are being developed under contract with the European Community as part of COMETT. These courses, further referred to as the Twente courses, are directed towards industry. However, the first also forms the basis for the "Real-Time and Parallel Systems" course offered in the Electrical Engineering Department at the University of Twente, Enschede, the Netherlands and the "In Parallel" course offered by the Open University in the Netherlands. Thus, the courses do have a wider application than is at first apparent. The relationships are shown in 1.

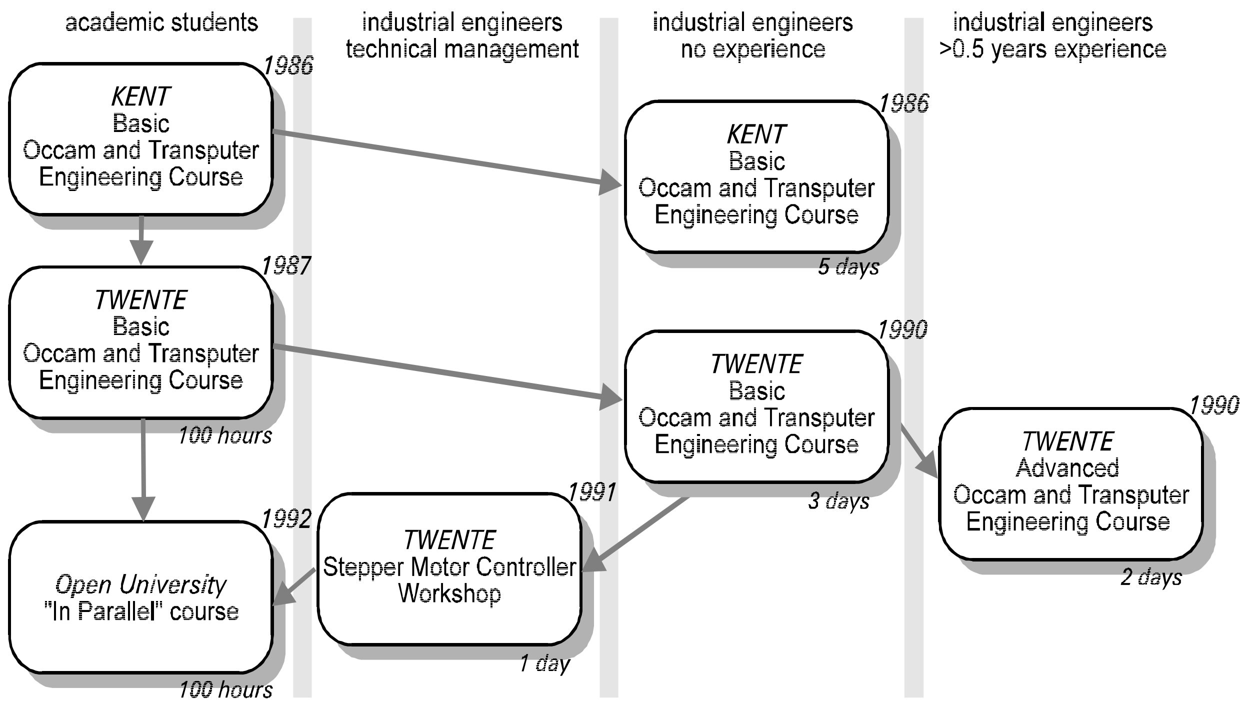
academic students
industrial engineers technical management
industrial engineers no experience
industrial engineers >0.5 years experience
1986
KENT Basic Occam and Transputer Engineering Course
1986
KENT Basic Occam and Transputer Engineering Course
5 days
1987
TWENTE Basic Occam and Transputer Engineering Course
100 hours
1990
TWENTE Basic Occam and Transputer Engineering Course
3 days
1990
TWENTE Advanced Occam and Transputer Engineering Course
2 days
1991
TWENTE Stepper Motor Controller Workshop
1 day
1992
Open University "In Parallel" course
100 hours

FIGURE 1: Taxonomy of the COMETT courses on transputers from Kent and Twente.

Three levels of education were distinghuished in the target group from industry:

- People with no experience in parallel processing, who just want sufficient information to base a decision on the application of transputers on.
- People who have decided to apply parallel processing, but who want to get a running start.
- People working for some time with transputers, but looking for specific knowledge on optimization and debugging.

Courses were developed for these levels of education.

3 Setup of the courses

The setup of the Twente courses is influenced by the properties of the target group. This resulted in the following general requirements for each course:

1. The courses should clearly show that parallel programming is not difficult. Instead, it is even easier than sequential programming.
2. Each course should have a clear structure, in which all pieces fit together. No sidetracking material that does not fit in the general philosophy may be introduced in the course. This material would at the least be ballast, at the worst even confuse the participants.
3. The parallel thinking is augmented by having the participants obtain hands–on experience by making exercises. These exercises are related directly to the theoretical material in the courses.
4. The participants should obtain a good working knowledge of the transputer and the supporting software tools. The knowledge obtained in the courses should be sufficient to get them started when trying the exercises for themselves back at the company.

As a result, the contents of the courses are more restricted than the academic courses. The general objectives for the courses are that they should give a practical theoretical background, i.e., the courses restrict themselves to giving a theoretical background in combination with examples, exercises and hands-on experience for that knowledge on parallel processing that is practical for people performing engineering with transputers. This means for example that:

– No taxonomy is given of parallel/multiprocessor hardware in general;
– No taxonomy of parallel processing programs is given, such as pipe lining, farming, geometric parallelism etc., instead only the basic building block approach is described;
– Only the Communicating Sequential Processes model of parallelism is discussed.

3.1 Stepper Motor Workshop

This workshop was created as a half day workshop to be offered in conjunction with conferences such as those of the World Occam and Transputer Users Group (WOTUG). Its goal is to demonstrate how parallellism provides both a unified design method and a way of increasing system performance. This is demonstrated by having the participants perform a hands-on experiment whose goal is the design and implementation of a parallel program that controls a stepper motor in real-time.

A participant in this workshop is not expected to be experienced in parallel programming or in programming in Occam. However, the participant is expected to have some programming experience. The workshop provides all of the basic materials and support to realize the parallel design and implementation of the controller in Occam. The development environment is a PC running MS-DOS with an add-in transputer board. The lecture section of the workshop is limited to a short introduction to Occam and the parallel building blocks that are available from the libraries. The Practical Portion of the Course Most of the workshop is spent with the participant receiving a lot of hands-on experience. Using basic building blocks and a library of routines that provide access to the stepper motor hardware, the participant designs a control program using standard design techniques. It is required that this be done using a set of communicating processes that run in parallel.

Controller testing is performed using a simulated stepper motor on the screen (also provided in a library). The interface to the simulated stepper motor is identical to the interface of the real stepper motor. The participant can switch between the simulated and the real-time environments by simply switching the name of the library call for the stepper motor process.

The participants work at their own pace. The practical portion of the workshop consists of six basic exercise plus three more for the more advanced participant. Each of the exercises builds on what has been learned in the previous ones. They are:

– Parallel Processes: In this exercise, the participant implements a loop that includes a set of parallel processes and displays information on the screen, using the stepper motor simulator display.
– Timer and Delay: The program from exercise one is modified so that the data that is output is slow enough that it can actually be viewed on the screen. In doing so, the participant is introduced to the concept of timers in Occam and the value of delay loops.

– Coil Control: In this exercise, the participant designs and builds a coil element with a specified initial state. Then a complete stepper motor controller is built.
– Variable Delay: The participant alters the stepper motor controller so that it has an adjustable and a variable speed. To do this exercise, the participant must learn about the ALT construct in Occam.
– Button Handler: In this exercise, the participant builds a "manual" interface to change the motor speed using the '+' and '-' keys.
– Real Motor Control: In this exercise, the online simulator is replaced by the real control cabinet and motor. The participant is rewarded by seeing his/her controller perform using actual hardware.
– Smooth Motor Control: In this exercise, the controller is modified so that the movement of the motor is smoother.
– Event Handling: The participant learns about Occam event handling by adding an emergency STOP button to his/her controller
– Bi-directional Motor Control: The controller is modified so that the motor can be rotated in either direction.

3.2 Basic course

The Basic Occam and Transputer Engineering course is a concentrated blend of lecture and hands-on experience in the design of parallel software for the transputer and in the use of the Occam language. The Twente course is offered in a three day format, whereas the Kent course has a five day format. However, the three day format runs about 12 1/2 hours a day while the five day format runs eight.) An extension of ANSI Standard C, PACT C, that provides parallel constructs similar to those of Occam, is also presented, and the participants have the opportunity to experiment with this language, as well.

The objective of this course is that the participants acquire technical knowledge of, insight into and practical experience in parallel system design using Occam and transputer networks. It is designed for a broad spectrum of participants from people who have had no prior exposure to Occam, to Occam users who want to learn how to exploit its power, to engineers of highperformance, high-security systems. Attendees are expected to have prior programming experience. Practical experience is provided using the Inmos Occam Toolset.

The course covers software engineering principles, process allocation techniques, real-time applications and various embedded and supercomputing issues. This course emphasizes the strengths, weaknesses and likely future developments of Occam and transputers. The focus in the course is on maximizing parallelism. Good software engineering principles are stressed through the use of block diagrams that show how individual parallel components communicate.

Lecture topics include the concepts of parallelism, software engineering issues, Occam and the transputer, the Occam model of parallelism, the Occam language, parallel system building, hierarchies of construction, new issues for the parallel system designer such as deadlock, livelock and non-determinism, buffered communication, multiplexing and demultiplexing channels, designing parallel software components for reusability and testability, and issues in security, performance, and real-time. The lecture material is liberally sprinkled with examples.

One of the strong points of the course is its practical sections. Typically, the participants work in pairs at their own pace. The group work helps participants to learn from each other as well as from the lectures and course notes. Each pair of participants is set up with their own system. They are provided with a working directory that contains the template files for each of the exercises. These templates illustrate good structured programming techniques. Each exercise builds on the experience the participant has gained from the prior exercises. The participants are required to create block diagrams of their programs before implementing them. Each exercise focuses on a particular concept:

1. *Thinking in parallel*: The participant is asked to design and implement a simple parallel system utilizing building blocks that have been presented in the lecture and that are available in a library. The emphasis is put on building small standard components with channels as their interface, creating a schematic diagram of the connection of these components and implementing this in a program[6][1][2]. The most elegant solution is that with the smallest and simplest components. Participants often are surprised by the simplicity of the components that can be used to solve the exercises.

2. *Deadlock*: People should get a good grasp of what deadlock is, what it is caused by, and how it can be detected and prevented. In these exercises, the participant is asked to format the output that is generated by the program and to suspend, resume and stop the output in response to specific keyboard input. Designing a test-rig. Dynamic modification of process activity, deadlock and graceful termination: In this exercise, the participant extends the keyboard handling of exercise 2. In response to specific keyboard input, particular processes must be dynamically modified. For example, if the input is a 'p', the subtraction in a process must be replaced by addition or the addition by a subtraction on a toggling basis. As there are several such modifications to be made, it is quite probable that deadlock will occur. In order to do this exercise in its entirety correctly, the participant must implement an overflow buffer and must learn to terminate processes gracefully. The latter means that ALL processes must be terminated not just the process that interfaces to the host[7].

   This set of exercises is structured in such a way that most participants will obtain a deadlock. Due to the order of the different exercises, a solution that fulfills the demands of the first exercise will almost certainly deadlock in one of the later exercises. If participants resolve all exercises without encountering a deadlock problem, the tutor will ask them to explain whether they noticed the deadlock and how they avoid it. However, most participants will run into a deadlock problem.
3. *Buffering*: Buffers introduce asynchronous behavior into a program, and thus are required in many cases. However, on a transputer buffers require an ALT construct, which is rather difficult to grasp. Therefore, the task in this exercise is to design and implement a suitable screen display for demonstrating the "Dining Philosophers" problem. Here, the participant's display process must not interfere with the simulation but rather must asynchronously and periodically report the status of the simulation. The problem here lies in the appropriate use of the timers of the transputer and the selection of an appropriate buffering and querying scheme. Shared memory solutions are prohibit, as these result in synchronization problems in practical systems. One of the main problems participants have is that they forget that processes are relatively cheap on a transputer. As a result, complex IF statements are used instead.
4. *I/O–sequential* versus *I/O–parallel* building blocks: With I/O–parallel building blocks deadlock can be prevented in actual systems[5]. In this exercise, the participant is asked to design and test a digital logic circuit, tricycle, with two inputs, clock and clear, and two outputs, bit.0 and bit.1, with the following behavior: Pulling the clear input low sets the output bits to 01. Then, each 'tick' on the clock steps the output bits through the sequence: 01, 10, 11, 01, ... which repeats indefinitely. Three problems are involved here: the learning of the difference between I/O–sequential and I/O–parallel problems, the synchronous behavior of the circuit as a whole, and the problems due to initialization and delay in I/O–parallel processes.

3.3 Advanced Course

The objectives of the two day Advanced Occam and Transputer Engineering course are to lead the participant deeper into the process of designing generic Occam procedures, to help them acquire a better understanding of the inner working of the transputer, in order to build faster, smaller and more efficient programs, and to learn how to use the post-mortem debugger. It is expected that the participant in this course has some experience in working with Occam and transputers. Practical experience is provided using the Transputer Development System.

Lecture topics include transputer architecture and processes, transputer assembly language, the mechanisms of channel communications, ALT inputs and their overhead, generic Occam programming, inner loop optimizations, parallelization and serialization speed-up methods, and double buffering and memory optimization.

The exercises include the how to use the debugger, the working of the Occam compiler and the properties of the generated code, evaluating the techniques presented in class by both timing code to which the techniques have been applied and applying the techniques presented to other algorithms. The algorithms used to demonstrate these concepts include matrix addition and multiplication as well as FFT calculations.

During the exercises on the use of the debugger and the generate code by the compiler the participant is asked to explain what happens during at least two of the exercises. This is possible due to the low participant to tutor ratio. In later exercises on size–invariant routines and optimization, an element of competition is introduced. In these exercises, the user is asked to optimize a matrix addition and a matrix multiplation procedure. By applying abbreviations and retypes, a substantial gain can be obtained. A final speed–up results from code reordering, i.e., the order in which integer and floating point instructions are performed can be optimized.

4 Comments from participants

Design of a Parallel Stepper Motor Controller in Occam
This workshop has been offered at two conferences. Most participants are able to finish at least seven of the exercises. The ninth is a real challenge and almost no one is able to complete it. A participant with a computer science background remarked on the clarity of presentation and the preparedness of the presenters. She commented about the amount she learned in just a half day, especially given her non-engineering background.
Only one notable negative comment was given, by a Danish participant: "Your workshop is much too difficult. I've been programming transputers for three years now, but what's this PAR construct?".

Basic Occam and Transputer Engineering
Most of the over 1000 participants in this course have indicated that the course had been very helpful to them, and that they would recommend it to colleagues. Some wished it could have been longer. Among the suggestions made for a longer course were the following: that additional time be devoted to more instruction and hands-on experience in the problems of mapping programs to networks of transputers, and that there be more discussion of practical applications. The participants have given high praise for the organization of the course, but some indicated that the presenter should provide the motivation behind the individual exercises. This would help the slower participants to decide on which exercises to concentrate their efforts.
While the course is specifically geared towards engineers, some participants come from other disciplines such as computer science. The course, while perhaps a bit more difficult for them because the use of electrical component as building blocks is not particularly natural for these participants, the approach and the results are useful. In order to learn to have a parallel mind set, such participants need to be weaned away from the construction of state machines to using parallel processes instead.
Most participants are well into the fourth exercise by the end of the course and several will have completed the fifth and have started the sixth. This was true also for a pair of participants at a recent offering of the course at the University of Twente who had NO prior programming experience. Although they had a lot of difficulty with the first exercise, they were well into the fourth exercise at the time the course ended.

Advanced Occam and Transputer Engineering
This course has been presented to over 100 participants in England and the Netherlands. The response to this course has also been positive. In the latest offering of the course, the class as a whole worked on an additional exercise that of solving a problem that one of the participants was facing at work. This deviation was well received and will likely be incorporated in future offerings of the course. There were two main suggestions for improvement: that the presenter should state the motivation behind the lecture topics and the exercises and that more lecture time should be spent on the debugger commands. Nevertheless, many participants have stated that they would like to take the course a second time.

5 Conclusions

The Training for Transputer Technologies series proved to be both popular and valuable. The hands-on approach with a low participant to tutor ratio seems to bring participants up to speed very quickly, even those with no prior programming experience. The use of parallel building blocks and the goal of maximizing parallelization while not producing the most efficient code does provide a platform for distributing the parallel processes over a network of transputers. The courses all encourage good software engineering practice including abstraction, generic processes, reuse and testability.

References

[1] Bakkers A.W.P. (1988), Application of parallel processing to robot control. *Proceedings of the First International Conference on Robotic Technologies TECHRO'88*, Bourgas, Bulgaria.

[2] Bakkers A.W.P. and J. van Amerongen (1990), Transputer based control of mechatronic systems. *Proceedings of the 10th IFAC World Congress*, Tallinn, Estonia.

[3] Hoare C.A.R. (1978), "Communicating Sequential Processes", *Communications of the ACM*, **21**(8).

[4] May D. and R. Shepherd (1987), "*The transputer implementation of occam*", Technical note 21, rep.nb. 72 TCH 021 00, Inmos Ltd., Bristol, UK.

[5] Welch P.H. (1987), "Emulating digital logic using transputer networks − very high parallelism = simplicity = performance", *Proceedings of the PARLE Conference*, Lecture Notes in Computer Science, Volume **258**, pp. 357−373, Springer−Verlag.

[6] Welch P.H. (1988), "An occam approach to transputer engineering", *Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications*, Pasadena, CA.

[7] Welch P.H. (1989), "Graceful termination − graceful resetting", *Proceedings of the 10th Occam User Group meeting*, Enschede, The Netherlands, pp. 310−317.