Architectures, Languages and Techniques
B. M. Cook (ed.)
IOS Press, 1999

Legacy of the transputer

Ruth IVIMEY-COOK
Senior Engineer,
ARM Ltd,
90 Fulbourn Road,
Cherry Hinton, Cambridge

Ruth.Ivimey-Cook@arm.com

Abstract. The Inmos transputer was more than a family of processor chips; it was a concept, a new way of looking at system design problems. In many ways that concept lives on in the hardware design houses of today, using macrocells and programmable logic. New Intellectual Property (IP) design houses now specialise in the market the transputer originally addressed, but in many cases the multi-threaded software written for that hardware is still designed and written using the techniques of the earlier sequential systems.
The paper discusses the original aims of the transputer as a system design component, how they have been addressed over the intervening decades and where we should be focussing our thoughts for the new millennium.

1. Introduction

The transputer concept was born in 1983[1] with the notion that a single chip computer could be considered a mere component of a system, not its core, and that indeed many such components might be used together. The name transputer was a concatenation of transistor and microcomputer; the intention was to indicate that this was a class of micro-computers with a similar nature to the then ubiquitous transistor; programmable components which could be mixed and matched as required, with few support components.

In many ways, both the concept and the initial implementation (the T414) were ahead of their time. The T414 was for several years one of the fastest generally available microprocessors. Its use of a 32-bit internal and external architecture, while not revolutionary, was still notable. Its speed and stack based architecture gave it an excellent interrupt response time. It was, however, the focus on the embedded systems market which was both its strength and its downfall. At the time, most processors used in embedded systems were 8 or even 4-bit devices. Many used or were based on popular 8-bit microprocessors of the day such as the Intel 8051[18] which sold in huge volumes throughout the period. The Inmos transputers entered the embedded marketplace with chip prices two orders of magnitude higher and, other than the 16-bit T212, using a costly 32-bit bus. The result was that despite a lot of interest and excitement, very few could afford it. Indeed, at the time many commentators were wondering whether a 32-bit bus was ever necessary; certainly there seemed to be no case for an embedded controller using one.

The market which eventually started using transputers in large numbers was in the relatively new arena of very high performance embedded systems, which encompassed applications as diverse as military sonar and missile systems, laser printers, fast network packet analysers and high performance databases. Because of its performance, many also tried to use it as a general purpose processor; the Atari Transputer Workstation and the Meiko Computing Surface were well known examples, and Parsytec sold many general purpose machines based on transputers. Although these more general systems were occasionally useful they often fell foul of chip design decisions favoring its use in embedded controllers. Many too found difficulty using occam – for a long time the only language supported by Inmos. A notable public success of this period included Quantel's Paintbox, a very high end image manipulation system. A useful summary of the transputer, occam and the use to which it was put was published by J. Edwards [5].

The high end embedded systems market has taken a long time to open up in a big way. Since 1997, the 32-bit ARM7[1] design has gained popularity in devices from mobile phones to highly integrated engine management systems, and together with its 32-bit rivals from Hitachi, Motorola and MIPS, have moved much of the embedded controller market away from 8-bit microcontrollers such as the 8051 and continue to take market share from the still very popular 16-bit 68000 and 80186[19].

The emergence of the ASSP[2] as ASICs[3] became more standardized, and the wider use of large macrocells in programmable logic devices has finally created the inexpensive, highly component based silicon marketplace the transputer foreshadowed. You are highly unlikely to ever see a package proclaiming to be an ARM7 CPU in a mobile phone, WebTV or engine management system; it will instead be buried inside an ASIC which also implements a memory controller, peripheral I/O interface and maybe even analog signal functions. Companies such as ARM Ltd. and MIPS are based on this marketplace, relying solely on selling Intellectual Property (IP) – designs and the knowledge to use them – rather than silicon. With system on a chip (SoC) designs, the ASIC is taken to its logical conclusion. Simple examples of SoC systems would be a pocket calculator or digital watch, but many more complex designs are being created too, some with several interacting CPU cores each with their own memory and peripherals.

The market has moved in another very relevant sense too. In the early '80s few products needed more power than a single processor. Advances in processor speed and power brought with them opportunities for more powerful products, but few designs tried to harness more than one processor. However, recently the design of a midrange car computer was rumoured to have 26 microprocessors scattered over the body of the car, communicating on an internal highly reliable serial bus. While this may seem like good news to those who appreciated the transputer, what we have lost is the software element with its rigorous approach to parallel system design: many of these systems are still implemented in C or even C++.

So as a concept the transputer is not dead. Its apparent death can be likened to the death of a visionary – the vision lives on. This paper will consider that vision, and its effects on us today.

2. Computing Components

The idea that a processing unit need not be the "main core" of a system is not new. Mainframe systems have for years now used satellite systems to process input and display results. The technology used in the initial transputers was not that special, either, but two things were:

- the level of integration – both of hardware and of the microcode driving it
- the application of formal techniques to the design of both the silicon and the instruction set

The latter point is important because the author believes those techniques steered the design of both hardware and language, shaping them to become the useful tools which resulted. With the first transputers, a complete computer took little more than the chip it was implemented on. That chip

[1] Around twice as powerful as the T805 for integer code

[2] Application Specific Standard Part – see Glossary

[3] Application Specific Integrated Circuit – see Glossary

had enough power to do an awful lot on its own, but if that was not enough a communication link, implemented on just three wires, were all that was needed to add another processor.

2.1 Requirements for Embedded Systems

The embedded computer systems market has changed along with the rest of the computer industry, but the essential requirements of the embedded marketplace are rather different. Embedded computer systems are simply those computer systems which have been incorporated into another product, and which typically perform only one, predefined task. They are said to be embedded because it is very often not obvious that there is a computer present – a good example is an electronic washing machine, or a lift. Embedded computers often use a class of operating system known as the real time operating system (RTOS), for which a wide choice exists.

Real time operating systems get their name from the fact that they are used as part of systems which must respond in a timely fashion to events in the real world. Burns and Wellings[9] point out that "*the correctness of a real time system depends not only on the logical result of the computation, but on the time that result is produced*". Real time systems are further divided into two groups: *hard* and *soft*. Hard real time systems partake in explicit deadlines; a given result must be available by a given time or it is useless and the system has failed. Often such deadlines are not only internally important to the system but life- or mission-critical. Soft real time systems may have deadlines, but many do not; in these, correct operation may be achieved without explicit reference to time. Such systems simply have to reliably run "fast enough". It is probably true that this division is not absolute. Some systems have deadlines which are more critical than others.

The embedded systems market places other requirements on computers. In many, there will be a need to reduce power and system size and be able to use unusual or varying supply voltages. Some are used in hazardous environments and may get very hot or cold. In many, overall system cost is critical; a few pence extra spent on components may become very important to the overall viability of the product. Lastly, time to market (the time between the initial product idea and the product in a customer's possession) has been important and is becoming ever more so. In some parts of the industry, a difference of a fortnight can now make or break a product.

2.2 The transputer and the occam language

The design of the transputer was very different to that of the standard processors of the day. The Intel 8051, for example, had a large instruction set, a fair set of registers and a relatively simple ALU. Peripherals included timers, parallel and serial I/O and some memory; outwardly, not very different to the T414. When you looked inside, however, the peripherals looked and behaved much as they would if off-chip. The assumption was that the system designer would create whatever software was required, and that was enough. The designers T414 made no such assumption. Four integrated communications links were supported not only by individual DMA engines, but by machine instructions. Those instructions would initiate the transfer and suspend the current thread of work, or task, until the communication was complete. The two built-in timers could be effectively shared, again using the task metaphor using lists of waiting tasks managed by the microcode, and the hardware interrupt signal behaved as a communication channel. This level of integration offered significant benefits in:

- reduced system design time;
- reduced system memory;
- reduced low level programming and testing.

But the Inmos designers carried on: using the same instructions communication could occur not only over the hardware links, but between tasks which shared same processor. As well as describing the tasks, scheduling was built in, and context switching was nearly instantaneous (even in current terms, 1µs is fast). Lastly, Inmos created a high level language (occam) which abstracted these features into a relatively portable[4] form which could be reasoned about, mathematically transformed and checked, providing high level of confidence that a program which compiled would also run correctly. This is something which C was, and still is, simply incapable of achieving.

So did it all work? The answer was very definitely yes. With a simple DRAM controller interface and no requirement for a boot ROM, processor PCBs could be extremely simple; it became common for two processors to fit on a credit card sized board, even without fancy surface mount technologies. Although the memory interface wasn't quite as simple for peripherals, video, serial, A/D and D/A cards with dedicated processors became plug and play components. Communications speed over the processor links was usually fast enough at 1Mbyte/s[5] not to be a problem, and links could often be doubled up for extra bandwidth.

Software design, too, was simple and effective. Occam, the language Inmos created[2,3], was based on research by C.A.R Hoare[4], who described a mathematical language called CSP. CSP describes the interactions of processes (things doing something) interacting through the exchange of events (signals or messages). In CSP, the internals of a process are opaque, as are the events. In occam, CSP processes are implemented as tasks, and events as messages communicated over one-to-one links. This mapping retains an extremely important feature of CSP: processes can be composed. That is if two processes α and β interact together and with some environment, the external behaviour can be modelled as a process γ. If β is replaced with another process implementing its external behaviour you also end up with γ as long as α and β "compose". Composition of processes is believed to be essential to the ability to effectively design systems. Consider how it would be if you had to use a particular finger to switch on a light – real world things do exhibit composition.

In their book[9], Burns and Wellings compare three languages in their usefulness at programming real time systems: C with POSIX extensions, Ada and occam. The C language was developed by B. Kerninghan and D. Ritchie out of their experiences with the earlier language "B"; it was designed to be a "high level assembler". It relies for its run time environment on libraries of functions; ANSI have defined a "standard C library", which is available for most compilers, but it is not necessary to use this. The POSIX threads library extends the C environment into simple multithreaded systems, but with little or no compiler checking or support.

Occam was developed to specifically support the development of the fine grained parallel processing environments supported by the transputer model and typical of those found in embedded systems. It also supports extensive checking of programs, threaded or not. One obvious deficiency of occam is that while its tasking model is very much more rigorous than that of C/POSIX, it does not support well the requirements of hard real time work, although some work has been done in this area to improve this[21,22]:

1. it is not possible to explicitly set hard deadlines for scheduling;
2. it is not possible to easily define the priority of a task.

While it would be good, few software systems or languages directly support (1). Absolute task priorities are however supported by almost every current RTOS. Priorities in occam may be indicated using the relative ordering of processes in a PRI PAR statement. This is fine until you wish to invoke a new process as part of the body of another (for example, using a procedure body to encapsulate a device driver). Because there is no global definition of the priority of a process in a PRI PAR, all the device driver can say is that the first process has a priority *at least* as high than

[4] Although occam has not been ported widely, this is not a particular fault in the language. However, see section 3.1 concerning the KRoC ports.

[5] OS Links ran at 10 or 20Mbits/s; newer DS Links at >100Mbit/s.

any other *it has created*. There is no guarantee that PRI PAR changes priority at all. Another subsystem may create what it considers to be low priority tasks which in actual fact have a higher global priority, without even being aware of this.

Ada grew out of the 1970’s US DoD review of software costs and languages[23]. The language which developed from that review was based on Pascal, PL/1 and Algol68 and entered general use in the early 1980s. Reviewed and extended in 1995, it includes a fairly coarse grained tasking model requiring significant source text to describe. One of the main aims of Ada was to allow the compiler to catch many common errors by providing the compiler with a very verbose language syntax and semantics. Based on its pattern of use it would appear to achieve this: much of the safety critical software written today is written in Ada, and this is only partly because the US Government used to mandate this for Government sponsored projects. However, the types of errors being detected are at the level of simple typing errors (common enough). It is not nearly as simple to formally prove Ada correct as it is occam.

There is a subset of Ada which must be used on human safety critical systems, and eliminates from the language some of the language elements thought to be more dangerous. Unfortunately, this includes all of the concurrent constructs; multitasking is considered by some to be inherently unsafe. As discussed elsewhere in relation to Java, this view is far from uncommon, although the author believes it to be incorrect.

2.3 Hardware macrocells

It was the very simplicity of the transputer design which was at the core of the “computing component” idea which was the transputer; the ability to just take a chip and use it, without the expense of designing and debugging large PCBs, or the problems of developing low level software to drive them. The same idea is behind the development of silicon macrocells. Macrocells are widely used in the silicon foundry business today to shortcut the incredible design costs, both money and time, involved in constructing chips with a complexity numbered in millions of transistors.

A macrocell is simply a block of logic gates, defined by an external interface which includes physical details such as where the wires emerge as well as temporal and silicon process restrictions. Many companies are involved in the construction of such silicon building blocks, some creating the software tools involved in their creation others selling space on silicon fabrication lines to build them. Importantly, there are also companies who sell nothing but the macrocells themselves. Technology in this area has advanced to such a level that processors such as the Zilog Z80, originally the product of many man-years of professional effort, can now be built by university students, and many more complex processors are now available as macrocells themselves.

The component computing ideas of the transputer have not been wholly subsumed by the new silicon based designs. While the macrocells do make certain things easier, enabling known working blocks of logic to be tied together with relatively little additional logic, the more useful parts of the transputer design have been lost: the unified communication architecture, the task metaphor, the very fast interrupt response time. For example, compare a T414 transputer, now nearly 15 years old, to the ARM7TDMI, one of the current generation’s faster responding processors and using silicon technology at least twice as good. Table 1 shows the ARM winning, but in the main because of a faster clock and the FIQ mode register bank, which means that a well-coded interrupt service routine need not save or load any registers. That advantage is mostly lost when IRQ mode is entered or when a task switch must occur. The figures for the 80286, which was fairly contemporary with the T414, show how advanced the transputer was for its time.

Table 1: Comparison of ARM7, T414 and 80286 processor response times (Best Case)

Processor	*Clock Cycle*	*Int. Latency*	*Task Switch*
T414	20MHz, 50ns	19 cyc, 0.95μs	19 cyc, 0.95μs
ARM7TDMI (FIQ)	40MHz, 25ns	4 cyc, 0.1μs	>40 cyc, >1μs
ARM7TDMI (IRQ)	40MHz, 25ns	13 cyc, 0.325μs	>40 cyc, >1μs
80286	8MHz		21μs

Notes:

1. Interrupt latency is taken to be the time from the interrupt assertion to the time when useful code can be executed. Memory is assumed to be zero wait state.
2. Figures for the T414 are taken from[1] p341.
3. The ARM is not using an MMU or cache, the IRQ service routines only save 5 registers, and the task switch is co-operative and only switching SVC mode. Because implementations differ, exact cycle counts of ARM instructions vary.
4. Figure for the 80286 are taken from [17] assuming the internal task switch instructions; interrupt latency was said to be similar. Scaling of the task switch figure for a 12MHz clock (much more common) produces a 14μs response time.

2.4 Operating Systems, Objects and Tasks

Programmers writing in occam very often found they were writing on “bare metal”. Initially there was no conventional operating system ported to the processor, and I believe many quickly found that in fact there was really little need for one. When the communications, memory management and task scheduling are performed in the language or by the processor, the typical remaining function of an OS is device management. When your devices are (or could be) distributed across several processors the obvious implementation of a device is simply a process responding to messages requesting some service. One of the few problems left to solve is how the device drivers are located; while in many cases it is acceptable to hard code this in the process configuration of a system, the occam community proposed many alternative strategies[6, 7, 8, 22, 32].

When your applications conform to the embedded application space for which the transputer was designed, implementing device drivers as simple processes in turn leads to a benefit: the driver can be tuned to implement essential functions of the application as efficiently as possible. For example, a graphics server the author wrote was extended to include an operation which drew a horizontal line formed of segments each of which was in a particular colour; hardly a common requirement, but for the application in question an extremely useful one. A typical occam program is thus composed of a number of client and server processes, which may be drawn from libraries or written for the application in question. The server processes implement device drivers and other higher level services such as TCP stacks, video encoding, database access or data compression. They are linked together with top level code which initialises the servers in the appropriate way and fires off the initial requests.

This structure is eminently object orientated; each process encapsulates data which describes an object; messages (real, in this model, rather than the method calls of C++) invoke an object to perform some action, possibly returning a result. A given object process can be extracted from its original surroundings and reused relatively easily, and should a new implementation become available, it could often be replaced with no changes to the surrounding code whatsoever. Importantly, the object can accept, defer and reject requests as it decides, depending on its internal state. This is a necessary requirement of composition [28, 29].

This modelling of objects in occam leads to an interesting corollary: implementing real world objects as occam processes leads to the conclusion that a given set of inputs to the real world object and the model should produce the same outputs. It becomes possible to exercise the real world object to collect real world test data, which can then be used to check the model directly. Testability is one of the big problems for the software engineering industry. As programs get larger and more complex, not only does it become impossible to exercise all possible states in the system, it becomes very hard to exercise all the code in each module. The ability to test modules in isolation with a reasonable hope that when in the system they will behave the same makes the tests easier to construct and the results easier to interpret.

2.5 Commercial Realities

We thus had hardware and software which was simple to plug together, with significant benefits to both the hardware and software sides of a project. These benefits were gained at a price. Few project managers wanted the pain of retraining personnel and translating source into a new, and very different looking programming language, even when the benefits were understood. Often those benefits were at best poorly understood and frequently system and language features were seen as mere syntactic sugar and dismissed. Inmos, as with many other innovative companies past and present, did not have the financial cushion to wait for the market to catch up and was finally taken over by SGS Thompson.

For a while, the high performance embedded markets for which the chips were designed were sufficient and the new process technologies used by SGS Thompson improved yield and speed, but as the delivery of faster versions slipped further and further, that market found other chips – the PowerPC, Alpha and MIPS among them. As technology advanced the previously rather limited Digital Signal Processors (DSP) started to become more general, and chips such as the Texas Instruments TMS32032 and TMS320C40 and more recently the Analog Devices SHARC found a ready market.

3. Birth and Rebirth

The Inmos transputer device range, which started with the T414 in 1983, was continued with a series: the 16-bit T212, T222 and T225. The 32 bit line was extended with the T425, which ran with a faster processor clock, more internal memory and also improved the instruction set and debug architecture. The T800, announced around 1987, included a formally verified IEEE[6] conformant FPU (quite rare for the period[7]) and was developed soon after the release of the T414. It was followed by the T801 and T805, introducing among other things the improved software debug also seen on th T425. The M212 was an early trial of an ASSP: an MFM disk interface controller version of the T212.

The T9000, intended as the next in the line, was announced with support for a significantly extended instruction set, a hardware implementation of virtual channels, superscalar performance and clock speeds which once again matched the competition, but for various reasons the chip was delayed, suffered badly from silicon bugs, and eventually canned years late having reached about a quarter of its intended performance.

Finally, the T400 (a 2 link T425 with reduced internal memory) and T450 (a T425 with extra internal memory and enhanced instructions) heralded the “official” end of the line of transputers. By the time the T450 (later known as the ST20) was in the field, interest in the transputer range as a whole was waning, and having already turned away from occam SGS Thompson announced “last orders” on the transputers themselves and gave the impression it was dropping the line entirely[8].

Other processors were moving on too, as shown in figure 1. Intel’s line of x86 processors was steadily growing, with the 80186 embedded system version gaining a lot of popularity, helped by

[6] ANSI-IEEE 754-1985

[7] Indeed, the author believes this is a unique achievement.

[8] “Gave the impression” because it would appear that the core, stripped of several links, seems to be available once again, under a different name.

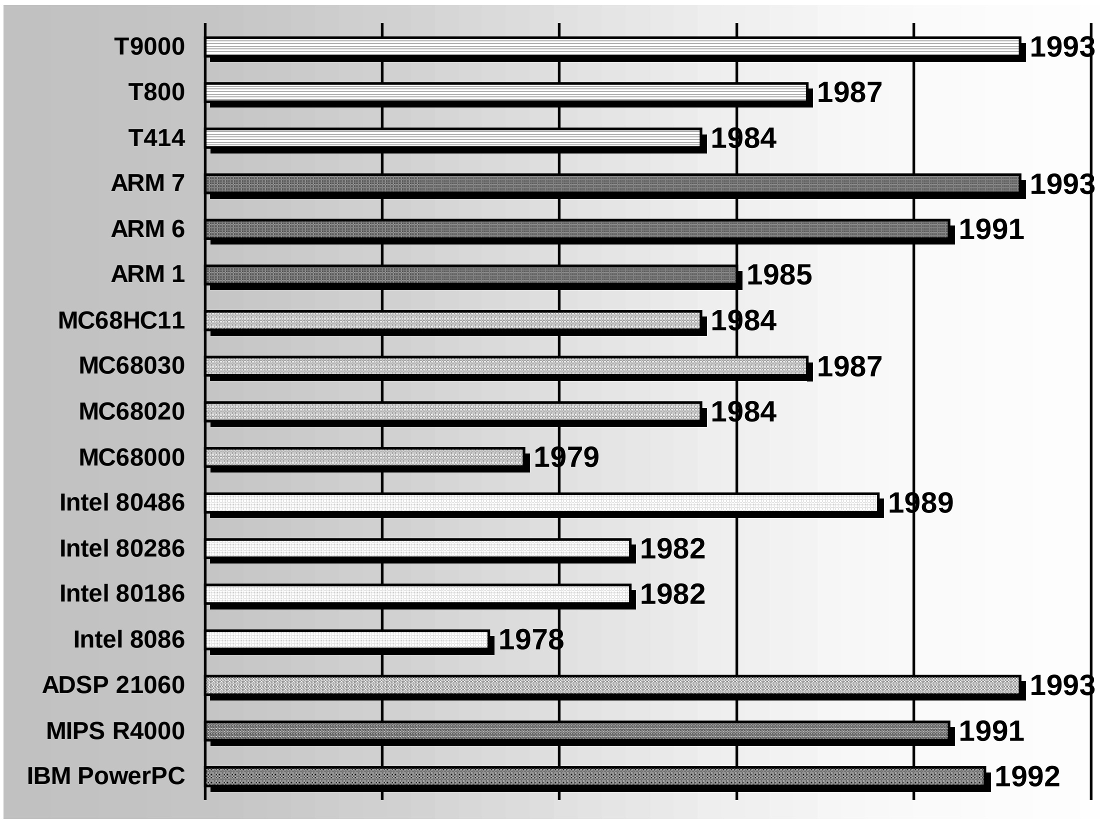

Figure 1: Approximate release dates of various Transputers and their competitors, past and present

the ready availability of compilers, support tools and chips produced for the burgeoning IBM PC market. Although the x86 range was not particularly powerful, they were quite capable of many of the tasks being asked of them. The Motorola 68000 series, having dropped significantly in price since its release and gained a number of specialised variants, also gained a lot design wins. Both the Intel and Motorola processors had one significant point in their favour – low price. Gained partly through volume and partly through in house fab lines, much of the market is very price sensitive.

Hitachi entered the market with it's Z80 compatible range of processors – the HD64180 being a significant advance on the ageing Z80, and have carried on with the SH series which in recent years have been very successful indeed. A number of new designs emerged from Zilog themselves, but the Z8000 and Z800 never caught on in a big way.

In the meantime, a community of academic and commercial developers had grown around the technology; user groups such as WoTUG, and NATUG grew and gained large followings. For some time, the community supported several conferences a year averaging 250 delegates each[9]. A lot of experience was gained in the use of the very fine grained parallelism which was offered by the transputer. An interesting observation was made that the programmers with a background in hardware design fared better with the design of these highly parallel systems than did those with a traditional computer science background. This experience has undoubtedly had its effect on the software community at large; a very large number of engineers have at least heard of the transputer, and quite a number of those people are aware of the capabilities of the device. What is a bit sad is that what is remembered best in the industry is the transputer's quirkiness – for example its use of an odd language as well as the rather unusual inclusion of fast communications links in preference to GPIO lines or an RS232 interface – rather than the benefits these things provided.

On the commercial front a lot of products were developed around the Inmos suggested TRAM

[9] For the period 1988 through 1991 WoTUG conferences alone ran to 250 delegates every 6 months. Transputing-91 attracted 450 paying delegates.

format. This was a PCB with a base unit size slightly longer than credit card size, which could be plugged into a motherboard with connectors along the short sides. Designs requiring a larger area could use multiple TRAM 'units'. The external connectors carried mostly a number of OS Link interfaces. Bringing out address or data bus would have been contrary to the basic principle of the transputer. The design was well received and many devices were put on TRAMS, including high resolution graphics cards, RS232 and RS432 serial interfaces along with the expected T4 or T8 CPU plus memory. Although at the beginning few single TRAM products appeared, by the late 1980s there were several designs including 2 or even four transputers on this quite small PCB.

3.1 Retargetting occam

The occam language, though hampered by the lack of compilers for other chips, waxed as real time systems developers discovered its expressive power. However, as the processors lost their speed advantage against other processors, people found them less and less practical, and the use of occam waned too.

Proponents of occam started trying to break the link between occam and the transputer the mid 1990s with initiatives to create a portable occam compiler. A dedicated Intel 80386 compiler[14] and the Southampton Portable occam Compiler (SPOC)[10] were some of the first of these to bear fruit. Later the Occam for All project resulted in a number of systems, including the Kent Retargetable occam Compiler (KRoC)[11] as well as several others[10,12,13]. SPOC made use of a compiler generation toolset to create a translator from occam to C, which could then be compiled with a standard C compiler. The result was quite usable, and not too inefficient. SPOC has been ported to several operating environments, including Sun SPARC, Intel PC and various DSPs. KRoC took another tack, making use of the then available ex-Inmos occam compiler: transputer assembler from this compiler is translated into native machine code for the intended target, which can then be assembled using native assemblers and linkers. KRoC ports now exist for Intel Pentium, Sun SPARC, DEC (Intel) Alpha, Motorola PowerPC, Motorola 68k, ARM and others, including some 8-bit microcontrollers.

The author believes that the efforts of the community have now created a viable base for the occam language and while the number of processor ports is growing, even if slowly, this can only improve. The raison d'être of occam is still just as true now as when the language was designed. Indeed, with many commercial operating systems and applications now being multi-threaded the reasons for using occam, or at least the ideas which occam embodies, are even more relevant now.

3.2 Applying CSP to Java

One application of the principles of occam is the JavaPP library[25] developed at the Universities of Kent at Canterbury and Twente, Enschede. Java includes support for multiple threads in its applications, using inheritance from a threads class to initialise and run threads. The Java threads model is based on C. Hoare's idea of monitors[27] which predates his development of CSP[4]. Hoare was unsatisfied with monitors in part because the way monitors synchronised did not lead to a concept of processes that composed, making the construction of systems very hard to reason about. JavaPP imposes on top of the Java threads implementation the CSP model of processes, including the required communication primitives[10].

Applications built using JavaPP automatically benefit from the clean mathematical and compositional base given by CSP. Designers and programmers do not have to be CSP experts, but they will notice that the behaviour of their components becomes more predictable. Race hazards,

[10] An obvious question: surely if JavaPP can use Java threads, the application can too. This is true; the point is that within JavaPP is a lot of very carefully thought out code implementing a model which is already known to work.

although not mechanically ruled out in the way that the occam compiler can rule them out, become much easier to control. The problems of deadlock, livelock and process starvation can (mostly) be tamed by following fairly simple guidelines[29]. It is interesting to note that Javasoft have recently started recommending that the threading in Java is not used simply because without a great deal of care it does not behave as expected; it seems reasonable to suggest this is because the underlying model – monitors – is not well-enough understood.

4. Multiprocessing: Modern Applications

Multiprocessing systems are now used in many different ways. The systems which are now being computerised are increasingly complex, and increasingly they are not possible for a human to readily comprehend the all at once. Consequently, modular systems, with readily defined interfaces are becoming more popular, as is the construction of even embedded software systems using very high level languages such as C++. This move is driven by the need by those implementing them to both comprehend the system being constructed and to test the result.

These more modular systems will, the author believes, power the drive towards greater use of the techniques pioneered in the transputer. Constructing applications and systems using threads libraries such as the POSIX pthreads or Win32 threads API is now much more common than it was even a decade ago, and is increasing as consumers demand easier to use, less modal systems. This use of threads libraries is making it easier for computer manufacturers to justify designing multiprocessor computers; in the past, very few applications could benefit from a multiprocessor design because they were solidly sequential. Crossing this bridge, as I believe we have, is an important point on the way towards a more general acceptance of the principles of concurrency. Another bridge crossed is the old idea that only a compiler should have to worry about parallel computation. This idea seems to have developed out of:

1. A general fear of concurrency, leading towards hopes that it can be “hidden away” in a compiler;
2. The fact that many supercomputers were vector processors, and vectorising compilers had been very successful at parallelising some types of problem.

The first point can be addressed neatly when a consideration of the problem of sequentialising a parallel system vs. parallelising a sequential system is made. The former is so simple that we can readily design very simple programs to do it; they are called schedulers. The latter is so complex it has taken decades for even the limited progress we have now. It can thus be concluded that we should in fact be writing highly parallel systems as our standard methodology, not writing sequential ones unless we are forced to do otherwise.

The slow but steady drop in the number of parallel vector supercomputers has led to a, sometimes forced, acceptance of other types of architecture. It is mostly from this domain that the work which has culminated in the Parallel Virtual Machines (PVM) and Bulk Synchronous Parallelism (BSP) parallel architectures, both of which seek to hide the implementation of the parallel machine from the application, with a consequent gain in application portability.

There are still bridges to cross, not least the preconception of many programmers that concurrent programming is intrinsically hard. This has come about from the bad experiences of the pioneers, who found out the hard way about deadlock, races and process starvation. Poorly conceived ideas about how processes should interact, and unreasonable hopes about the possible benefits of “going parallel” only add to this massive education requirement.

For all that, the author believes it will happen. The car computer mentioned in the introduction had a large number of processors arranged in a distributed memory architecture. Such systems can be developed using traditional techniques, but those techniques are ill equipped to deal with the systems level problems which result.

4.1 The new transputers

There is a lot of debate in the transputer community about what a new transputer should ideally be, indeed whether there is any longer a need for a dedicated processor. A great deal has now been achieved on modern RISC processors by modelling the methods used by the transputer in the machine language of other processors, even when those processors are using a heavyweight operating system such as Unix. These advances have been incorporated into the current generation of occam compilers, and hold great promise for those who merely want or need a good parallel execution environment. Some of the flaws of this path currently include comparative compilation inefficiency compared to traditional languages (for which good optimising compilers exist), and a comparatively opaque structure.

On the other hand, as we have seen earlier the Inmos transputers gained a great deal of speed and power from their integration, and a number of people would like to see a modern transputer. The current “favourite” for such a part is something based around an ARM core, using integrated DS Link[31] (IEEE 1355) hardware. Such an approach, if taken simplistically, would obviously lack the instruction set modifications the transputer benefited from, but by using the ARM coprocessor architecture a certain amount of integration could be achieved. A “real” transputer, in the Inmos style, would require core macrocell changes and would probably be prohibitively expensive. Other ideas favour the further development of the hardware / software codesign, outlined below, which promises to provide massive benefits for those who are willing to use it.

At least some of the transputer ideas have been picked up by other manufacturers. Both Texas Instruments and Analog Devices have incorporated transputer-link style communications devices onto their DSP engines: first, the TI 320C40 and then the ADSP 21060 (SHARC) series of DSP processors (see Figure 1) used a byte-wide link with associated DMA channels, which have proved very popular. A KRoC port was attempted to the SHARC, with mixed success due to the very different nature of the transputer and SHARC instructions[20]. However, these differences are being addressed and an improved version is under development.

4.2 Hardware compilation and hardware description languages

In parallel with the work of retargetting occam to other software platforms efforts are also being made in the hardware compilation of occam programs. This work is currently creating designs for programmable logic devices such as FPGAs, but there is little to stop such designs being implemented in more dedicated logic. Work in this field has progressed on two lines: the theoretical basis of occam in the form of CSP can be extended to account for the passage of time, and the construction of two compilers targeted at digital logic rather than processor instructions. Hardware-CSP is largely the construction of A. Lawrence, who took the existing CSP and extended it in two directions: adding the notion of process priority and prioritised choice[15], and then building on that by adding the notion of time[16]

There are at least two groups working in this field; I. Page at Oxford has produced Handel-C[26], a version of occam, dressed up in C syntax, with some crucial extensions to allow the specification of bit-widths for standard data-types and a new notion of single-cycle synchronised parallel assignment. Handel-C is now part of a commercial product. R. Peel and B. Cook have worked on creating an occam translator with the aim of taking one of more arbitrary sections of occam code and creating hardware to run it; the remaining software (if any) can then invoke the hardware actions. This work would seem to be advancing well.

5. Summary

The intention of this paper was to summarise the impact of the idea and implementation of the transputer on the software and hardware engineering industry of today. We have looked at the basis of the transputer – the elements of simplicity and in integration that made it a useful and very powerful processor in its own right – and compared them to the current methods used in silicon production: the production of large macrocells using design tools. We noted here that while the transputer's simplicity and of component computing have, to come extent, been carried forward here, there is still little integration of the hardware and the rest of the system involved in this process.

We have also looked at the occam language which Inmos designed with the transputer to provide high level access to these facilities. An examination of the languages used in the embedded systems domain shows that while occam is not a widely used language, both C and Ada make life hard when describing fine grained parallelism, and C at least provides almost no safety when using any kind of parallel task model. Concurrency in Java is better, in that at least the target bytecode is typesafe, and there is a built in threading model, but users have had some difficulty in applying it, demonstrated in part by JavaSoft dropping a thread safe GUI interface[33]. As those developing JavaPP have shown, the CSP concurrency model reaps significant benefits.

Finally, work currently in progress is enabling the use of occam as a hardware description language; the hope is that arbitrary sections of program source can be compiled into hardware and linked seamlessly and automatically with other sections running on a traditional processor. It has been found that the CSP basis of occam makes this a lot easier than for almost any other current language.

The author concludes by recasting words originally applied to Pascal, but which now apply with great force to both occam and the transputer:

"They are great advances on their successors"

One day, soon, the world will realize it needs them. Fortunately, the knowledge has been preserved.

6. Glossary

ALU	The Arithmetic and Logic Unit of a processor; that part which carries out data manipulation such as addition, ANDing and shifting. Some processors contain more than one ALU, and some use an ALU for address calculation in addition to data (operand) calculation.
ASIC	Application Specific Integrated Circuit. A chip which has been designed and fabricated for a specific task and for a specific customer. It would be usual for the customer to provide whatever technical support (documentation, etc) was required. It is possible that an ASIC can become an ASSP (qv.)
ASSP	Application Specific Standard Part. A chip which is being made available to many customers, complete with vendor technical support.
BSP	Bulk Synchronous Parallelism. A technique in which a set of parallel processes advance through a computation one step at a time, in lock step synchronisation.
CSP	Communicating Sequential Processes. A formalism used to describe the interactions of processes (whether real world or not) which interact using shared events.

Deadlock	A software failure mode in which two (or more) interacting processes simultaneously need a resource allocated to the other. Also known as deadly embrace. Deadlock may involve many processes which form a ring.
DMA	Direct Memory Access, a method of transferring data between locations in memory or between memoy and an I/O device without the use of processor generated addresses.
FPU	Floating Point Unit; that part of a processor which performs floating point arithmetic.
IEEE 1355	The IEEE standard which describes the DS Link protocol, which was one of the successes in the development of the T9000. Like the OS Links it is byte-serial, and supports speeds in excess of 200Mbit/s. See the 1355 Association: http://www.1355-association.org/
IP	Intellectual Property. Ideas, methods and designs which can be traded by themselves, rather than their implementations. Often applied to companies such as ARM and MIPS, who sell CPU core designs to silicon foundries.
JavaPP	A parallel processing library for Java based on occam. See http://www.cs.bris.ac.uk/~alan/javapp.html and http://www.rt.el.utwente.nl/javapp/
JCSP	A parallel processing library for Java based on CSP. See http://www.hensa.ac.uk/parallel/languages/
Livelock	A software failure mode in which although activity is occurring, progress in completing the assigned work is no longer being made.
NATUG	The North American Transputer User Group. See. http://multi.ece.usu.edu/natug/
PVM	Parallel Virtual Machine. A library which abstracts a virtual parallel machine from real parallel hardware.
RTOS	Real Time Operating System; a class of computer operating system in which time is an element in the success of the system. See section 2.1.
SHARC	An Analog Devices term: Super Harvard Architecture RISC Procesor, used to describe their ADSP 21060 range of processors.
SoC	System on a Chip; an ASIC which incorporates all, or very nearly all, of the [digital] components of a system on one piece of silicon.
Starvation	(of a process) A software failure mode in which one or more processes are starved of work or data through the unintended dominance of other processes. Occasionally this is benign, causing only system inefficiency, but more often it results in apparently inexplicable failure.
TRAM	Transputer Module. A design of a generic link PC board.
Transputer	The name for a class of processors which integrate the cpu with several communications links. Pioneered by Inmos in the early ‘80s some other manufacturers have also used the ideas since.
WoTUG	The World occam and Transputer User Group, formerly the Occam User Group. See http:/www.hensa.ac.uk//parallel/groups/wotug/

7. References

[1] Inmos Ltd. The Transputer Databook. Second Edition, 1989, Inmos Document 72 TRN 203 01.

[2] Inmos Ltd. The occam 2 Reference Manual. Inmos 1988, pub. Prentice-Hall.

[3] SGS Thompson Microelectronics Ltd. The occam 2.1 Reference Manual. 1995, available at http://www.hensa.ac.uk/parallel/occam/documentation/

[4] C. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985

[5] Janet Edwards and Phillip Lawson. The Advancement of Transputers and Occam. Proceedings of WoTUG 14, IOS Press, 1991.

[6] Debbage M, Hill M, Nicole, D. The Virtual Channel Router. Transputer Communications 1, 1993

[7] R. Huang and M. Morgan. A Distributed I/O Communication Protocol for a Network of Transputers. Proceedings of WoTUG 18, IOS Press, 1995

[8] S. Harrison and C. Brown. WEAVE – A System for Dynamic Configuration of Virtual Links. Proceedings of WoTUG 20, IOS Press, 1997

[9] A. Burns and A. Wellings. Real Time Systems and Programming Languages (2nd Ed). Addison Wesley, 1997.

[10] M. Debbage, M. Hill, S. Wykes, D. Nicole. Southampton's Portable occam Compiler. Proceedings of WoTUG 17, IOS Press, 1994

[11] D. C. Wood, P.H. Welch. The Kent Retargettable occam Compiler. Proceedings of WoTUG 19, IOS Press, 1996

[12] T. Sheen, A.Allen, A. Ripke, S. Woo. oc-X: an optimising multiprocessor occam system for the PowerPC. Proceedings of WoTUG 21, IOS Press, 1998

[13] S. Kalogeropoulos. Developing an Optimising Compiler for occam. Proceedings of WoTUG 21, IOS Press, 1998

[14] M.D Poole. An Implementation of occam 2 Targeted to the 80386, etc. WoTUG News No. 18, 1993

[15] A. Lawrence. Extending CSP. Proceedings of WoTUG 21, IOS Press, 1998.

[16] A. Lawrence. HSCP: Extending CSP for Codesign and Shared memory. Proceedings of WoTUG 21, IOS Press, 1998.

[17] P. Wells. The 80286 Microprocessor. BYTE Magazine November 1984.

[18] H. Boyet and R. Katz. The 8051 One-Chip Microcomputer. BYTE Magazine December 1982.

[19] T. Zingale. Intel's 80186. BYTE Magazine April 1983.

[20] G. Otten, M. Schwirtz, R. Bruis, J. Broenik, W. Bakkers. Implementation of KroC on Analog Devices "SHARC" DSP. Proceedings of WoTUG 19, IOS Press, 1996.

[21] E. Ploeg, A Sunter, A. Bakkers and H. Roebbers. Dedicated Multi-priority Scheduling. Proceedings of WoTUG 17, IOS Press, 1994.

[22] P. Welch. Multi-priority Scheduling for Transputer based real time control. Proc WoTUG 16, IOS Press, 1993.

[23] W. Whittaker. The US Department of Defence common high order language effort. ACM SIGPLAN Notices 13(2), 19-29.

[24] A. Baker and K. Milner. A Process Migration Harness for Dynamic Load Balancing. Proc WoTUG 14, IOS Press, 1991

[25] P. Welch. Java Threads in the light of occam/CSP. Proc WoTUG 21, IOS Press, 1998.

[26] M. Aubury, I. Page, D. Plunkett, M. Sauer and J. Saul. Advanced Silicon Prototyping in a Reconfigurable Environment. Proceedings of WoTUG 21, IOS Press, 1998.

[27] C. Hoare. Monitors: An Operating System Structuring Concept. Communications of the ACM 7(10): pp549:557, October 1974.

[28] C. Hoare. Algebra and Models. Proceedings of the ACM SIGSOFT '93 Symposium on the Foundations of Software Engineering. 1993. pp1:8.

[29] J. Martin and P. Welch. A Design Strategy for Deadlock-Free Concurrent Systems. Transputer Communications vol. 3, 1996.

[30] G. Hilderink, J. Broenink, W. Vervoort, A. Bakkers. Communicating Java Threads. Proceedings of WoTUG 20, IOS Press, 1997.

[31] IEEE Std 1355-1995, IEEE Standard for Heterogeneous InterConnect (HIC) (Low Cost, Low Latency Scalable Interconnect for Parallel System Construction). IEEE Computer Society, 1995.

[32] P. Welch and D. Wood. Higher Levels of Process Synchronisation. Proceedings of WoTUG 20, IOS Press, 1997.

[33] See Javasoft web page "Threads and Swing" available at http://java.sun.com/docs/books/tutorial/uiswing/

overview/threads.html#rule, [Feb 19th 1999]