occam for all

Case for Support

1 INTRODUCTION

The multi-processing language occam provides a strong formal basis for the secure and efficient development of high-performance parallel applications. Industrial experience over the past eight years (limited to transputer-based hardware) has amply demonstrated its benefits for the engineering of scalable computing applications across a wide spectrum of problem area. Methodologies and supporting tools for specific areas (e.g. modelling, real-time control, databases, hardware design...) have been developed and remain the subject of vigorous and high-quality research (chiefly UK led).

This is a proposal to build a set of architecturally neutral tools, based on occam, that support the design, implementation, optimisation and maintenance of high-performance applications. Systems constructed with these tools will be portable across all current and currently foreseen (MIMD) parallel computing platforms, where they will operate with high levels of efficiency.

This programme is partly motivated by fears shared by all partners in the consortium – particularly the industrial partners – that the engineering and commercial benefits previously enjoyed through the use of occam on transputer-based platforms will be denied to them in the future *unless* the language is 'opened' to the wider parallel computing community. The positive motivation arises from occam being the *only* parallel-processing language in widespread industrial and academic use whose semantics is sufficiently well-defined, simple and powerful to play this unifying role.

The rest of this proposal has the following structure. Section 2 introduces the partners in this consortium and builds the case for the strategic importance of occam in parallel computing. Section 3 lists and justifies the set of tools that is the basis for the work-programme. Section 4 explains and details the work-programme priorities, the allocation of work amongst all the partners and the mechanisms for industrial and academic exploitation of the results. Time-scales and costs of the project are detailed in the RG-2 application form.

2 INDUSTRIAL AND ACADEMIC PARTNERS

Our consortium consists of industrial *end-users*, technology *suppliers* and academic *researchers*. For about ten years (on average), all have been successfully involved with parallel computing in their business and/or research and teaching activities, for which it now forms an essential element.

Our work programme is driven by the needs of our *end-user* partners, British Aerospace (BAe) and GEC-Marconi Avionics (GMAv).

Both their letters of support describe the success they have achieved working with occam (confidence, performance and cost). BAe, in particular, are keen to preserve the technical benefits they have obtained through the use of occam for parallel systems. Both companies wish to retain and exploit their parallel computing expertise based on occam. Our work programme will both enable this and open occam as a significant tool for portable parallel computing for all levels and types of industry.

INMOS are the developers of occam and will play a crucial role in our project, donating compiler sources and technical advice. We will work with the Architecture Group (headed by David May) that developed the latest occam3 definition and are currently responsible for the new Chameleon architecture. We will also work with the INMOS Software Products Group who are responsible for compiler development. Our project will engage in primary research into implementation strategies for the novel communication paradigms proposed in occam3, with the active participation of both these groups.

Formal Systems (Europe) Limited market a suite of analysis and transformation tools for occam systems and CSP designs. These tools can be significantly improved by exploiting developments in the language, but this work may not be commercially worthwhile unless the language is opened to the general parallel computing community (e.g. through the porting efforts proposed in our project). Formal Systems will provide technical advice on all work packages and are in a position to offer commercial support and marketing for products resulting from our work.

The principal academic investigator for this project is Professor Welch (University of Kent), who has a long established record of innovative work in this field. Co-investigator is Dr Cook (University of Keele) who has special expertise in compiler technology as well as parallel computing. His experimental occam-to-C compiler currently produces the fastest executing occam code on any processor. This is not a full occam implementation but will be used for rapid prototyping. The later work packages will apply ideas from the recent research of both these investigators [12, 3]. Both will give a significant portion of their time to this project and are backed by active parallel research groups within their departments.

3 TOOLS AND WORK-PACKAGES

3.1 The language as a tool

Language shapes our thoughts – it is our fundamental tool for formulating ideas, communicating them and applying them to our advantage. Language must evolve to capture higher levels of abstraction as they are discovered and found to be useful – but language must remain as simple and precise as possible lest it contributes more to our problems then our solutions.

occam was designed in the early 1980s to meet the challenge of parallel computing. It is built upon important theoretical insights gained during the previous decade (primarily those of CSP [6] and CCS [9]), whose significance has been confirmed by the subsequent decade of practical application. occam provides and *enforces* a rich set of abstractions for parallel processes, synchronisation and communication. Its compilers incorporate sophisticated analysis tools to ensure conformity with those abstractions. These tools are only made possible through the clean semantics of occam and have no equivalents for parallel extensions to standard imperative languages (such as C and FORTRAN), even though those extensions may target similar paradigms to those of occam. The fact that occam is an astonishingly *small* language is a significant help in creating these tools.

A key principle is the *abbreviation* mechanism of occam, that enables tight control over the aliasing of names to objects. occam uses this control to eliminate the aliasing of updatable objects, thereby greatly simplifying the semantics of assignment, bringing this into line with our engineering intuition. This yields immediate benefits for the safe and efficient implementation of parameter passing, but the major reward lies in the enabling of data and channel usage checking for parallel programs. This latter guarantees that non-determinism is *not* introduced simply through the use of parallelism. It results in system properties that are stable against the vagaries of the ordering in which processes are scheduled and against the number and performance characteristics of the processors on which processes happen to be allocated. When seeking tools to assist the production of applications that are portable across parallel architectures, this uniquely qualifies occam as the (imperative) language on which they should be based.

occam offers a range of other tools for parallel, distributed, real-time and safety-critical computing: *explicit control* over the level of non-determinism; *dynamically sized and dynamically structured* message-passing that is checked at compile-time (*note:* most other systems support messages only as arrays of bytes whose structure is open to mis-construction by the sender and mis-interpretation by the receiver); *a high-level view* of time, time-outs, interrupts and priority scheduling; *really lightweight processes* whose scheduling can be managed with a context-switch overhead of less than 10 micro-seconds on most modern processors (and less than 1 micro-second on a transputer); *a formal denotational semantics*, based on the (timed) traces-failures-divergences model of CSP.

At present, these tools are restricted for use only on transputer-based hardware. Our first group of work packages builds on preliminary efforts by the academic partners and others to open these tools for other processors.

3.2 *(Work Package I)* SPOC extensions and optimisation

The Southampton Portable Occam Compiler (SPOC) [4] compiles occam into C. It was developed under the Esprit GP-MIMD (P5404) project and has recently been placed in the public domain. Applying the Gnu C compiler with full optimisation to the SPOC output produces reasonably fast code on the standard range of (single processor) UNIX workstation. SPOC is not based on any INMOS compiler sources, being a complete re-implementation using the GMD public domain compiler-building tools and a sophisticated novel algorithm for parallel usage checking. This work-package is to perform the following tasks:

(a) extend SPOC to support external processes (such as the user keyboard) properly. At present, if a SPOC process is waiting for user input, *all* processes have to wait! This is incorrect and prevents interactive user control of running systems – this is not acceptable. [*Note:* SPOC does produce code that can interact with the INMOS *IServer* running as a separate UNIX process, provided the user has access to INMOS proprietary libraries. This leads to correct I/O-synchronisation but also results in significant run-time overheads, even when no I/O is taking place.]

(b) optimise SPOC to produce less finely-grained process scheduling – currently SPOC performs a context-switch after *every* occam statement:

(c) extend SPOC to support multiple processors. Initial targets are the Parsytec PowerGC and Ethernet-connected workstations. This will require the development and implementation of a simple configuration language. [Note that a good technical solution to task (a) above will be essential for this.]

3.3 *(Work Package II)* Native-code generators

An alternative approach to the rapid re-targeting of occam compilers is to emulate the transputer architecture within the target processor and reuse the INMOS compiler (almost) intact.

An instance of this has been demonstrated by Michael Poole (an Independent Consultant), who has implemented a full occam2 compiler for 386/486 PCs [11] and is now marketing and supporting this commercially. This benchmarks faster than SPOC for straight serial computation and for context-switching. Our work-package comprises the following tasks:

(a) using occam2 compiler sources donated by our INMOS partner, modify the final stage of code-generation so as to yield code for a demonstration range of modern high-performance processors. These processors will be directly related to the commercial and academic needs of consortium members for parallel systems and will include at least three from the set {SPARC, PowerPC, DEC-alpha, TI C41, R3000, ...}. [Note: each processor will need to be supported by a software micro-kernel to supply the functionality provided by the transputer's micro-coded scheduler.]

(b) investigate the generation of re-targeted codes directly from the transputer-assembler output from the INMOS compiler. This will remove the need for access to INMOS sources and will enable immediate tracking of the latest INMOS compiler and language developments (e.g. the optimising back-end and occam2.5 language extension, both due for release this year). Experience with sub-item (a) will be necessary for this investigation, especially with regard to the elimination of processor-specific dependencies (e.g. *endian-ness*) currently introduced at earlier stages of code-generation.

(c) design and produce a tool to automate significant stages in the production of re-targeting transformers for new processor architectures. The produced transformers will be either bound into the INMOS compiler or, preferably, stand-alone if sub-item (b) shows this to be feasible.

(d) extend these systems to support multiple processors – with the same target systems as work-package I(c).

3.4 *(Work Package III)* Evaluation

This is to evaluate the performance of the systems produced under work packages I and II. Industry standard benchmarks (for serial languages) will be run. Parallel benchmarks that are specifically geared to the needs of our industrial *end-user* partners will be developed and tested. Some of these benchmarks will be representative codes of 'industrial strength' that will be lent to this project by our industrial partners to support this evaluation. This package will demand close and continuous collaboration between all members of the consortium throughout the project duration.

3.5 Binding new tools into the language

When a new idea (design method, paradigm, ...) has been discovered, found to be useful outside its immediate region of discovery and matured into common use, it needs supporting to ensure correct application by naive users and and it needs efficient/portable implementation. The most effective means of support is to bind a suitable abstraction for the idea into the language. Then, the language user cannot misuse the concept and the language compiler has the necessary information from which to generate the most optimum code. For example, this has happened repeatedly in the development of structured, object oriented and parallel programming.

The designers of occam2 restricted themselves to capturing the correct *fundamental* abstractions for practical *parallel* computing (i.e. processes, structured and synchronised message-passing, explicit non-determinism, priorities, real-time and hardware interfacing).

In the period 1990-92 and in consultation with the occam user community (both academic and industrial), INMOS defined occam3[1] – an upwards compatible extension to occam2. Apart from introducing a full set of user-definable data-structures (e.g. records and tagged unions), occam3 built on the experiences of occam2 users to provide explicit high-level support for client-server systems, remote procedure calls, channel-structures, objects (that remain secure in parallel systems) and libraries. This has generated considerable interest and discussion in the literature (e.g. [8]).

However, INMOS have resourced the implementation of only a modest subset called occam2.5, which is scheduled for release later this year. occam2.5 introduces records and user-defined types and tidies up the 'awkward bits' that have emerged from seven years practice with occam2. INMOS have announced no plans for implementing any of the higher-level tools implied by the full occam3 definition.

3.6 *(Work Package IV)* Language extensions as tools

This comprises the following tasks:

(a) upgrade the enhanced SPOC and native-code generating compilers from Work Packages I and II to support the new occam2.5 standard. [*Note:* if we have been successful with the approach being researched in Work Package II (b), this tracking will be automatic.]

(b) research and develop efficient implementation strategies for the new high-level tools defined in occam3. We will concentrate on the communications support needed for securely shared channel and data structures that underpin the safe distribution of client-server systems and remote procedures calls. Prototype solutions will be bound into the SPOC and native-code generators. At the same time, we will review the occam3 definition from the point of view of both users and implementors.

(c) increase the portability of the occam3 language definition. One example of this is to enable the user-specification of time granularity. Currently this is a machine-dependent implementation issue (that causes no problem since there are only transputer targets!). For real-time code to be portable across architectures supplying hardware timers running at differing speeds, occam timers need to abstracted from that hardware. Other portability problems are described in [3]. Language modifications to enhance such portability will be proposed and prototyped.

(d) all outputs from items (a), (b) and (c) will be evaluated. Item (b) will be especially useful to our INMOS partner for their commercial occam3 implementation for the transputer.

3.7 Development of new tools outside the language

Before new ideas have been developed to the state where their usefulness has been broadly recognised and accepted, tools may still need to be built to make them easier to apply. In our case, these tools will be *dependent* on the semantic properties of the occam language for their success, but need not (yet) be built *into* the language itself.

An example is ensuring the absence of deadlock in parallel systems. *occam*'s low-level synchronised message-passing, whilst guaranteeing that data cannot be lost through scheduling accidents, allows the possibility of deadlock. Its elimination is therefore a design issue, for which tool support will be helpful.

3.8 *(Work Package V)* Deadlock-free design rule checkers

In [12], two high-level design paradigms (*I/O-PAR* and *Client-Server*) are presented that guarantee freedom from deadlock for synchronised communication regimes (both continuous and irregular data-flow). The paradigms are based upon a notion of 'synchronisation classes' for processes that are closed under certain forms of parallel composition. Checking for deadlock freedom devolves to checking that the base processes belong to the correct classes and that the composition rules are observed. The complexity of this checking is at worst O(n^2), where n is the number of *processes* in the system, as opposed to O(n^2s^2), where s is the (average) number of *states* in each process. The latter would be required for an arbitrary parallel design. The automated checking of these design rules is therefore highly practical. The *I/O-PAR* and *Client-Server* paradigms cover most data-parallel and irregular process control applications. This work package comprises the following tasks:

(a) Research the automated checking of the design rules for deadlock-freedom for *I/O-PAR* and *Client-Server* parallel systems and for hybrids comprising both paradigms. This will require the front-end of either the SPOC or INMOS occam compiler for syntactic and semantic analysis and the building of a prototype checker that will work on the resulting parse trees.

(b) Investigate the relaxation of some of the rules given in [12] to allow a wider class of design to be checked.

(c) Make proposals for binding these design rules into the language. [*Note:* the occam3 support for client-server systems still allows deadlock into designs.]

4 PRIORITIES, ALLOCATION AND EXPLOITATION

4.1 Portability and elegance are not enough

From the point of view of *end-users*, it is not sufficient just to have tools that enable the correct design and implementation of applications that are portable between differing parallel architectures. If the level of overhead introduced by those tools to achieve portability is too high, they will be ignored in favour of an application-specific environment that extracts the best performance from the machine and the costs of porting will be accepted. This is already happening, for example, regarding the (lack of) performance of PVM on the Cray T3D.

Fortunately, occam-based tools for parallel applications offer performance alongside portability, clarity and security. Reasons for the last three properties were outlined in section 3.1. Performance follows from those same reasons, together with the total integration of parallel and serial computing concepts (including explicit message-passing and shared-memory) into the *language*. By having these ideas built into the language (rather than implemented through special system libraries), high-level information is presented to the compiler about the parallelism of the application that can be used to optimise the code produced for whatever parallel hardware we happen to be targeting. These optimisations are not easy to achieve by reverse-engineering general-purpose, but low level, 'parallel' system calls from a serial language in an attempt to gain the necessary understanding of the parallelism in the design.

4.2 The strategic importance of occam

The parallel concepts of occam are architecturally neutral of the platforms on which its programs are executed. They can be implemented with high levels of efficiency both on distributed and (virtual) shared memory MIMD systems – or, for ultimate performance, on custom/customisable silicon [10]. The formal semantics of occam [5] offer the prospect of 'closing the gap' between software and hardware systems [7] and of forming the backbone of a suite of formal tools that will integrate the development and verification of systems from high-level specification down to transistors – taking in general-purpose parallel computing on the way.

occam has no rival technologies competing for this role. Its loss as a widely-used and commercially supported tool would therefore be seriously damaging to our industrial and academic futures. Unless occam is opened up and made efficient for all architectures, this loss is very likely. Our Work Packages I-IV address this 'opening up'. They involve a substantial amount of work that can no longer be funded on the backs of other programmes. They specifically address the fundamental aims of this PSTPA programme. Neither INMOS nor any commercial software vendor (that we have contacted) is in a position to allocate resources to such work as a purely internal project.

4.3 Priorities, allocation and management of work

Therefore, we shall give highest priority to getting occam ported and efficient on to a wide range of architectures – this will secure the base of this technology. Without this base, building exciting tools on top of occam semantics will be an academically interesting task, but will have limited practical use. On the other hand, securing this base releases a host of existing analysis tools that are *necessary* for portable parallel applications and *cannot* be built on top of standard languages such as C or FORTRAN (for example, see the difficulties of ensuring parallel 'disjointness' in Brinch-Hansen's SuperPascal [2]).

The bulk of the work will be undertaken by the academic partners. Kent will concentrate on packages II (native-code generator), IV (tools as language extensions) and V (deadlock analysis tools). Keele will concentrate on packages I (SPOC enhancements), III (evaluation) and IV. British Aerospace and GEC-Marconi Avionics will concentrate on the evaluation package (including the supply of 'industrial strength' source codes) and will advise on packages IV and V.

INMOS will donate the compiler sources necessary for the production of native-code generators (II) and will advise on packages IV and V. Our research into implementation strategies for full occam3 (IV) is directly relevant to their future plans for transputer compilers and will be actively involved. [*Note:* UKC is already an alpha-test site for the INMOS occam2.5 compiler for the transputer.] Formal Systems (Europe) Limited will also advise on all work-packages, giving specific assistance on the INMOS compiler sources (on which they have gained extensive experience in the development of their own software tools).

For the modest support that we understand is available to PSTPA projects, we do not promise to complete all the Work Packages defined earlier (which represent a coherent plan for a continuing project). However, with 1.5 persons working full-time for two years, we shall complete packages I, II and III and make significant progress on IV and V. The programme of work will be managed by the University of Kent, with a formal Technical Committee meeting quarterly to review and plan progress. Each consortium member will be represented on this committee. Management will be kept as light as possible with normal technical exchanges between partners being conducted electronically. One full-time Research Fellow will be based in Kent for this project, with a further half-time post at Keele. We are requesting a small upgrade in memory and disk capacity plus X-terminals to support these people.

4.4 Exploitation of results

We will place all our tools in the public domain and actively promote them. Formal Systems will have first-refusal on taking over commercial support and marketting of all products from this project. An agreement will be made concerning royalty payments to the academic partners from any commercial exploitation of this work. There will be a stream of papers documenting all results which will be of a standard for acceptance by relevant (refereed) conferences and journals. Our products will open `occam` to a much wider community of user, will generate much wider academic research and commercial application and will help preserve the achievements of occam through its natural role of enabling safe, efficient and portable parallel computing for all.

References

[1] Geoff Barrett. occam3 reference manual. Technical report, INMOS Limited, Bristol, BS12 4SQ, ENGLAND, March 1992.

[2] P. Brinch-Hansen. The programming language SuperPascal. *Software Practice and Experience*, 24(5):467–484, May 1994.

[3] B.M. Cook. Some issues concerning the portabilty of occam programs. In Reinhard Grebe et al., editors, *Transputer Applications and Systems*, volume 2, pages 1170–1180, Amsterdam, September 1993. TTC, IOS Press. ISBN: 90-5199-140-1.

[4] M. Debbage, M. Hill, S. Wykes, and D. Nicole. Southampton's portable occam compiler (SPOC). In R. Miles and A. Chalmers, editors, *Progress In Transputer and occam Research*, number 17, pages 40–55, Amsterdam, April 1994. WoTUG, IOS Press. ISBN 90 5199 163 0.

[5] M.H. Goldsmith, A.W. Roscoe, and B.G.O. Scott. Denotational semantics for occam2. *Transputer Communications*, 2(1):25–67, March 1994. ISSN: 1070-454X.

[6] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[7] C.A.R. Hoare and I. Page. Hardware and software : The closing gap. *Transputer Communications*, 2(2):69–92, June 1994. ISSN: 1070-454X.

[8] Jon Kerridge. Using occam3 to build large parallel systems. *Transputer Communications*, 2(1):1–24, March 1994. ISSN 1070-454X.

[9] Robin Milner. A calculus for communication systems. Technical report, Laboratory for the Foundations of Computer Science, Edinburgh University, 1986.

[10] I. Page. Automatic design and implementation of microprocessors. In R. Miles and A. Chalmers, editors, *Progress In Transputer and occam Research*, number 17, pages 190–204, Amsterdam, April 1994. WoTUG, IOS Press. ISBN 90 5199 163 0.

[11] Michael Poole. An implementation of occam2 targetted to 80386, etc. *Transputer Communications*, 1(1):xiv–xv (WoTUG News No. 18), August 1993. ISSN 1070-454X.

[12] P.H. Welch, G. Justo, and C. Willcock. High-level paradigms for deadlock-free high-performance systems. In Reinhard Grebe et al., editors, *Transputer Applications and Systems*, volume 2, pages 981–1004, Amsterdam, September 1993. TTC, IOS Press. ISBN: 90-5199-140-1.