Figure 1: The Bundled Data Interface Protocol

Following Sutherland's approach, the AMULET group at the University of Manchester, U.K., have designed and implemented AMULET1, an asynchronous version of the ARM RISC processor as part of the ESPRIT OMI-MAP project (figure 2) [5].

2 Modelling Asynchronous Architectures with Occam

The recent research activity in the area of asynchronous systems has pointed to the need for suitable techniques for modelling and simulating them. Several notations and techniques have been suggested for this purpose [7]. CSP [6] in particular has attracted the interest of many researchers due to the strong relationship between its semantics (synchronous, unbuffered interprocess communication) and the behaviour of asynchronous systems [10]. To support the design of AMULET1, a methodology for using occam [8], a CSP based, parallel distributed language, for rapidly building executable models of asynchronous architectures at the Register Transfer Level has been developed [15] [16] [17]. Using this methodology, an asynchronous architecture is modelled as a set of concurrent occam processes which communicate by exchanging timestamped messages. The processes are entirely data-driven, and self scheduled. Two channels are used for the synchronization of communicating processes, one for Request/data and one for the Acknowledgement signal. To describe the nondeterministic behaviour of arbiters, the occam ALT statement is used. Following this approach, **occarm** , an occam model of the AMULET1 processor has been developed. **Occarm** consists of a hierarchy of occam processes, has the

Figure 3: Occarm Top Level Process Graph

machine, which has been developed at the University of Manchester to support the parallel simulation of computer architectures (figure 4). Two of the four links from each transputer of the T-Rack (*link0* and *link1*) are permanently hardwired to form a processor chain known as the *necklace*. The off-necklace links(*link2* and *link3*) may be connected by means of a crossbar switch, into a configuration which is appropriate for the code being executed. The crossbar switch is built using twenty six INMOS C004 switch chips housed on two boards (*S1* and *S2*). The T-Rack is hosted by a Sun 3/160 workstation containing a Tadpole Transputer Board which acts as the "root" node of the transputer network. The existence of the necklace limits the set of processor graphs which may be implemented to those which possess a Hamiltonean Cycle [11]; the route taken by the Hamiltonean Cycle through the network corresponds to the position of the necklace in the switched network implementation.

5 Monitoring

Monitoring the runtime behaviour of the simulation model and collecting information regarding the characteristics of the simulated system is one of the main objectives of the simulation process. Monitoring is essential for the testing and performance evaluation of the simulated system as well as for the debugging of both, the simulated system and the simulation model. The inherent properties of distributed asynchronous systems, such as occam architectural models, make monitoring a difficult and complicated issue for which

Figure 5: Event Traces for Debugging

sequential techniques are insufficient; these properties include the multiple threads of control, the difficulty to get snapshots of the system, the non-deterministic behaviour, the intrusiveness of the monitoring system and the need to cope with a vast amount of monitoring data generated from multiple sources [12].

5.1 Monitoring Occarm

The simulation of an asynchronous architecture has two main objectives, namely the testing and debugging of the architecture, and the evaluation of the architecture's performance.

5.1.1 Debugging

For the debugging of the architecture (as well as the simulation model) it is necessary to monitor both the flow of control and the flow of data in each of the different occam processes in the model; this is achieved by collecting traces regarding the *execution* and *data events* of the processes respectively (figure 5a,b). Furthermore, for the detection of deadlocks, it is essential to know the state of the processes in the system when the deadlock occurred. For this purpose, the *parallelity events* need to be monitored (figure 5c). These will appear in pairs, one for the sending and one for the receiving process. The absence of one parallelity event from a pair in the final trace indicates the occurrence of a deadlock.

In occarm, monitoring is performed by means of a parallel network of occam monitoring processes (known as *reactive processes* [12]), one for each of the top level processes (the *active processes*) of the **occarm** model. Monitoring messages are issued to the reactive processes in an event driven fashion. *Probes*, in the form of a procedure call, have been inserted (manually) in the source occam code of the active processes. Each time the procedure that implements the probe is called, it constructs the corresponding monitoring message and sends it to the monitoring process. Since probes will be invoked in different parallel sections of the active process, several monitoring messages are issued simultaneously. Thus, for the communication between an active and its reactive process a channel array is used (*monitor*, see figure 6). The monitoring process acts as a multiplexor, employing an ALT construct to gather the messages issued by the corresponding active process on the channel array; in order to reduce the effects of the ALT bottleneck, the channel array is buffered in order to decouple the processes involved.

The monitoring data are generated in a copious volume and their transportation can have a significant negative impact on both the computational resources and the communication network of the distributed system. Occam monitoring processes support two *transport strategies*, namely *immediate transport*, in which the monitoring data are transported as soon as they are generated and *store and unload*, whereby the data are stored in a buffer before they are transported. The latter makes use of a circular buffer to store only the recent history of the active process. If no monitoring message arrives for a user-defined time interval (i.e. in the case of deadlock), the monitoring process flushes the contents of the history buffer. The store and unload transport option is particularly useful as in most, if not all, cases the most recent history of the processes is sufficient to identify the cause of errors or deadlocks. Since the monitoring messages do not propagate further into the system, the communication overhead of

Figure 7: The Single transputer Environment of Occarm

at the output side). Stall situations refer to the input side of the pipeline. They may be detected by examining the delay between the sending of a Request event (R_{in}) to the pipeline and the issuing of the corresponding Acknowledgement signal (Ack_{in}) by the first register of the pipeline: a stall situation has occurred if $timestamp(R_{in}) < timestamp(Ack_{in})$. The duration of the stall is $timestamp(Ack_{in}) - timestamp(R_{in})$; clearly the minimum stall period is equal to the propagation delay of the first register in the pipeline.

Contrary to the debugging traces which are sent immediately to the corresponding monitoring process, the type and quantity of data that concern the performance characteristics of the architecture permit active **occarm** processes to calculate and store them locally; this eliminates the extra communication overhead that their transport would impose. The stored values are unloaded by the **occarm** control processes upon termination of the simulation.

6 The Simulator Environment

The single-transputer simulator environment is depicted in figure 7.

The Memory process models the memory control logic of the processor; the memory itself is implemented as a binary file to achieve compatibility with the existing ARM development environment [2]. The

Figure 9: Modified Occarm Top Level Process Graph

particular process. Merging two columns together effectively adds one more level of abstraction to the process hierarchy, assigning the corresponding processes to the same processor and forcing the two channels to share the same link.

7.1 Balancing the Workload

The criterion adopted for the selection of the level of the **occarm** process hierarchy whose each process has at most four neighbours is the maximization of processor utilization; namely, to occupy as many processors as possible.

Following this criterion the merges presented in figure 8 have been applied to occarm, deriving as a result the alternative graph of figure 9; this graph represents the lowest level in the process hierarchy that satisfies the four-link-per-transputer limitation.

7.1.1 Balancing the Communication Load

The new top level **occarm** process graph possesses more than one Hamiltonean cycle, thus allowing an equivalent number of possible mappings on the T-Rack.

For the selection of the appropriate mapping, the criterion which has been followed is to balance the communication load. In the T-Rack the communication performance of a hardwired link is approximately double that of a switched link (1.72 and 0.87 Mbytes per second respectively). Consequently,the objectives of the communication load balancing policy are to use as few switched links as possible, and to place onto

Figure 10: Occarm Graph Mappings

7.1.3 The Generic Simulator Node

Figure 12 depicts a generic node of the distributed implementation of the simulator. Typically this will include a number of active processes together with the corresponding monitoring modules. Extra multiplexing/demultiplexing processes are included to allow the sharing of the transputer links; to prevent deadlock situations (which for example might occur if one of the transputer links is blocked by a message destined for a particular process, while this process is blocked waiting for a message that may follow the former on the link), extra buffering has been incorporated into the the demultiplexing modules (i.e. the distributor process).

Figure 12: The Generic Simulator Node

Model	Elapsed Time (minutes)
Occarm (Single)	1.72
Asim	1.48

Benchmark: Dhrystone (1 loop)

Table 2: Asim versus Occarm (Single Transputer Implementation)

Transport Policy	Elapsed Time (minutes)	
	Occarm Single	Occarm Multi
Tracing Off	1.72	1.02
Store and Unload	4.22	1.87
Immediate Transport	9.75	7.21

Benchmark: Dhrystone (1 loop)

Table 3: Performance of Occarm

8 Performance

For the verification and evaluation of both the AMULET1 processor and the **occarm** model, the *Dhrystone* synthetic benchmark has been used [18]. Within the single transputer configuration (i.e. on a single 20MHz, T414 transputer), **occarm** requires on average 1.72 minutes to execute one Dhrystone loop when no monitoring traces are generated (table 2); this figure is smaller than that achieved by an equivalent sequential simulator written in ASIM (the ARM's in-house simulation language), by a factor of 1.16, when the latter executes on an IPX Sun workstation. This is a reasonable and expected performance, for the execution of the **occarm** processes on a single transputer is performed not in a parallel but rather in a time sharing fashion and the large number of processes in the model (more than 120) makes the context switching overhead in the transputer significant.

Table 3 presents the performance of **occarm** for both, its single and multiple transputer configurations and for the different policies employed for the transportation of monitoring data. When no monitoring traces are generated, the distribution of **occarm** on to the seven transputers of the T-Rack yields a speedup of 1.69. The "store and unload" transport policy allows a speedup of 2.26 to be achieved since in this mode of operation, the performance of occarm on a single transputer drops by a factor of 2.45 as opposed to 1.83 in the multi transputer implementation. This difference in the performance drop may be attributed to the fact that the activation of the monitoring processes severely increases the frequency and, consequently, the overhead of context switching on the single transputer. The distribution of the monitoring processes onto multiple transputers alleviates this phenomenon as the context switching overhead is also distributed.

When the "immediate transport" policy is employed, the performance of both the single and multiple transputer configurations of **occarm** drops dramatically allowing a speedup of only 1.35. This behaviour may be attributed to the operation of I/O process. I/O process acts as a multiplexor for messages arriving from both the Memory and the Monitoring processes. This introduces a major bottleneck in the system, which imposes the ultimate limit in the performance of the simulator. The large number of monitoring messages generated by the "immediate transport" policy occupy a large proportion of I/O process activity, thus reducing the rate that instructions and data are supplied to the model; as a consequence, the processes of the model remain idle for substantial periods.

The low speedups achieved by the distribution of **occarm** onto the multiple transputers of the T-Rack may be attributed to a number of factors related to the characteristics of both, the simulated architecture and the machine that hosts the simulator.

- Amdahl's law [1] specifies that the maximum possible speedup depends on the inherent parallelism of the executed system which may be potentially exploited. In the case of AMULET1, the requirement for instruction compatibility with the synchronous ARM, has resulted in an asynchronous design with a very complex pipeline structure and, indicatively, very limited parallelism. The performance of AMULET1 itself is only 70% of the performance of the synchronous ARM.

- Asynchronous architectures are communication bound systems and therefore the efficiency of the communication system is crucial. The complex irregular interconnection pattern of AMULET1's functional modules and the extra multiplexing/demultiplexing processes required to cope with the connectivity constrains of the Transputer and the T-Rack introduce major bottlenecks in the system that severely reduce the communication efficiency.

9 Conclusions

Asynchronous logic promises to provide solutions to problems such as clock skew, power efficiency, performance and modularity on VLSI design and is currently experiencing a resurgence of interest. AMULET1, being the first asynchronous implementation of a complex commercial RISC architecture, has demonstrated the feasibility and merits of the asynchronous approach. Occam is particularly suitable for the specification and description of asynchronous systems. Its parallel distributed nature offers the potential for high simulation performance, but introduces a number of design and implementation problems, common in parallel asynchronous programs. This paper has discussed the solutions and techniques developed to address these issues in connection with **occarm**, the occam model of AMULET1.

Acknowledgements

For this work Georgios Theodoropoulos has been supported by the *Mpakalas Foundation*, Athens, Greece, under Grant No. 466/21121992.

References

[1] Amdahl, G. M., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", Proceedings AFIPS 1967 Sping Joint Computer Conference, Atlantic City, April 1967, pp 483-485.

[2] ARM Ltd, Fulbourn Road, Cherry Hinton, Cambridge, CB1 4JN, England.

[3] Capon, P.C., "ParSiFal: A Parallel Simulation Facility", IEE Colloquium: The Transputer: Applications and Case Studies, IEE Digest, 1986/91.

[4] Brunvand, E. and Sproull, R., "Translating Concurrent Communicating Programs into delay-Insensitive Circuits", Technical Report CMU-CS-89-126, Carnegie Mellon University, April 1989.

[5] Furber, S. B., et al., "A Micropipelined ARM", Proceedings of VLSI '93 (Best Paper Award), September 1993, pp. 5.4.1-5.5.8.

[6] Hoare, C.A.R., "Communicating Sequential Processes", Communications of the ACM, Vol. 21, Number 8, August 1978, pp 666-677.

[7] Hauck, S., "Asynchronous Design Methodologies: An Overview". Technical Report UW-CSE-93-05-07. Department of Computer Science, University of Washington, April 1993.

[8] Inmos Ltd, "Occam Programming Manual", Prentice Hall, 1984.

[9] Inmos Ltd, "Networks Routers and Transputers", IOS Press, 1993.

[10] Martin, A. J., "Compiling Communicating Processes into Delay-Insensitive VLSI Circuits" Distributed Computing, 1, 4, April 1986, pp 226-234.

[11] Murta, A., "Tools for the Automated Configuration of a Transputer Network", MSc Thesis, University of Manchester, October 1987.

[12] Riek, M., Tourancheau, B., Vigouroux, X. F., "Monitoring of Distributed Memory Multicomputer Programs", Technical Report UT-CS-93-204, University of Tenessee, October 1993.

[13] Mead, C. and Conway, L., "Introduction to VLSI Circuits", Addison Wesley, 1980, Chapter 7, pp 218-254.

[14] Sutherland, I., "Micropipelines", Communications of the ACM, Vol. 32, Number 6, June 1989, pp 720-738.

[15] Theodoropoulos, G., Woods J.V., "Building Parallel Distributed Models for Asynchronous Computer Architectures", Proceedings of the World Transputer Congress 1994, Como, September 1994, pp 285-301.

[16] Theodoropoulos, G., Woods J.V., "Distributed Simulation of Asynchronous Computer Architectures: The Program Driven Conservative Approach", Proceedings of the European Simulation Symposium 1994, Istanbul, October 1994, pp 230-234.

[17] Theodoropoulos, G., Woods J.V., "Analysing the Timing Error in Distributed Simulations of Asynchronous Computer Architectures", Eurosim Congress '95, Vienna, Austria, September 1995, to appear.

[18] Weicker, R. P., "Dhrystone, A Synthetic Systems Programming Benchmark", Communications of the ACM, 27, 10, October 1984, pp. 1013-1030.