PARALLEL PROCESSING: A SAFER OPTION FOR REAL-TIME CONTROL SOFTWARE

C.I. Birkinshaw, P.R. Croll, D.G. Marriott, P.A. Nixon

The University of Sheffield, UK

Keywords: Parallel Processing, Real-Time Control, Formal Methods, Standards, Safety-Related, Software Engineering

ABSTRACT

Software is increasingly being used to control applications demanding high performance and high reliability. Modern parallel processing technologies can offer systems designers many advances towards satisfying these requirements. This paper considers whether the current software safety standards, and the methods they advocate, are sufficiently mature to reach this objective. A particular example of a parallel development method derived from formal mathematics is explored. The aim is to identify both a pragmatic and intuitive development method suitable to control systems applications. This method should permit a fair degree of automation of a control system's behaviour analysis necessary to establish a safe parallel implementation.

1. INTRODUCTION

Digital computers are increasingly being used for software control of complex systems. Consequently more emphasis is being placed on the safety aspects of these computers and the constraints that they must work under. In recent literature this has given rise to the term *safety-critical computing* [Red–93].

The purpose of this paper is to show parallel processing paradigms to be a mature solution to safety-critical computing systems, which can improve the reliability, safety, and performance of real-time control applications. The control community has embraced the flexibility and performance of parallel processing, for example transputers and occam [Irw-92], in a wide range of applications. Yet there still remains a significant need for the introduction of the formal theories which underpin these solutions. By utilising these formalisms, the control engineer can gain more confidence in the software developed.

A brief discussion of the application of parallel processing in real-time control leads into a short résumé of the various industry standards and their implications for control engineers. The paper concentrates on the formal techniques which are available to the software designer for the development of parallel control systems. A comprehensive review of the maturity and applicability of these techniques to the design of safe, parallel real-time systems is given. The emphasis will be on *push button* formal methods by considering the automated tools available to support the mathematics. This is considered as the only practical way of applying formal techniques [Aus–93, Cro-91], given the current level of expertise.

Livelock and Deadlock are two examples of common design faults in parallel processing systems [Bir-93]. These symptoms are caused by a lack of understanding of the behaviour of the systems by the engineers who build them, and are not the result of some "emergent behaviour" that appears by virtue of it being parallel. Once this myth has been dismissed, it becomes clear that what is needed is a set of tools and formalisms that aid understanding of parallel systems.

Good engineering practice has been acknowledged by Bennet [Ben–93] to be sadly lacking in software design. However, techniques for the construction of robust control applications have been demonstrated that do not require exhaustive formal proofs [Wel–93, Ame-93]. We can increase the standards of our designs by using tools to enforce good engineering practice; these are techniques already available to us, but would benefit from consolidation into Computer Assisted Software

CONTROL '94, 21–24 March 1994. Conference Publication No. 389, © IEE 1994.

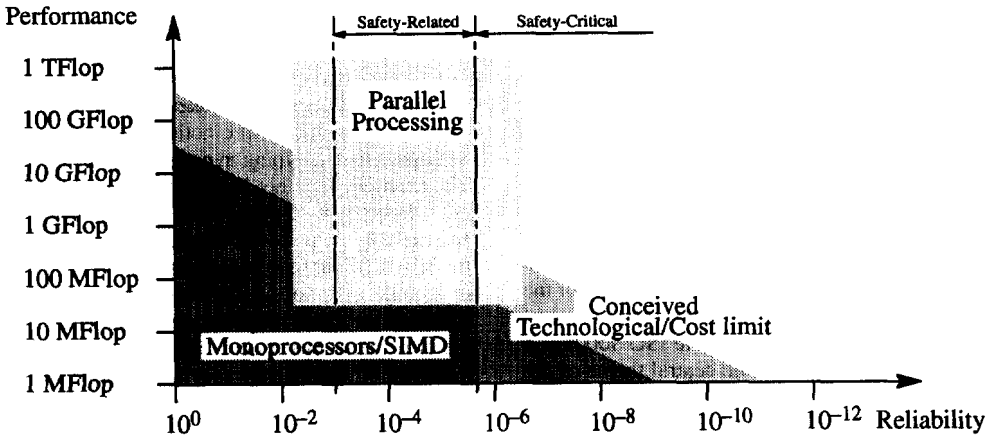

Figure 1. *Performance versus reliability of safety systems.*

Engineering (CASE) technology. Secondly, we can build on this by adding hide-away formal mathematics in CASE, which allow rigorous checks and proofs to be made. Application of the second set of techniques would be much easier to achieve if the first set was imposed on the designer through current CASE technology.

2. PARALLEL PROCESSING

Parallel processing offers the control engineer the high performance expected from modern real-time systems. It also offers the possibility of fault tolerance through redundant components. It is also argued that parallel processing offers new avenues for increasing the reliability of high performance control software, see Figure 1.

Parallel processing invites a simpler design route. The world is an inherently parallel place and the decomposition of a parallel problem lends itself naturally to a parallel implementation, avoiding the complexity which evolves from forcing a sequential solution [Wel-90]. Each parallel process is simple and self contained which aids analysis. Performance may be scaled by adding extra hardware instead of optimising the code, and this hardware also leads to increased reliability, introducing the possibility of fault tolerance through redundant hardware. These properties of parallel systems constitute the main argument for the use of parallel solutions in control systems.

3. STANDARDS: THE EMERGING TRENDS

There are a wide variety of standards bodies throughout the world concerned with software development. It is to these standards systems designers will turn for guidance when choosing a method for developing their system. This section briefly outlines four of the plethora of standards. It is apparent that the standards authorities are moving towards certification as means of quality assurance. The implication being, that the control engineer must be aware of these standards and the techniques they advocate The standards presented are IEC 65A, ISO 9000, MOD 0055, and HSE-87; these cover the main range of general standards applicable to software safety.

3.1. Quality assurance

The ISO 9000 standard does not directly address safety-related computer control systems, but does provide a framework for applying quality assurance within software development. There is a strong belief within the industry that this should be one of the minimum requirements for any software development, and compliance with ISO 9000 is required by the IEC 65A standards for all levels of software development.

3.2. Software for industrial safety-related systems

IEC 65A is a suite of standards that provides a classification of software integrity levels

– from high risk to normal. At the normal level, ISO 9000 compliance is sufficient. High risk software requires a formal specification and application of correctness proofs. This standard explicitly mentions CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM and Z as suitable methods.

3.3. HSE guidelines

The UK Health and Safety Executive (HSE) has produced its own set of guidelines for the design of safety–related computer systems, and suggests BS5750 (UK equivalent to ISO 9000) as a minimum requirement, but the central theme of the guidelines is diversity through a number of independent elements to provide sufficient redundancy to guard against failures.

3.4. Ministry of Defence equipment

UK defence standard 00-55 is based on the Ministry of Defence's procurement lifecycle. It specifies requirements for all safety-critical defence equipment. This is twinned with standard 00-56, which is aimed at a standardised approach for the whole system, including its interface with the environment. The standard extends to tools and support software used in developing, testing, certifying and maintaining safety-critical systems through all the phases of the software lifeycle. Formal specifications and proofs are required for all critical components.

4. FORMAL METHODS FOR PARALLEL PROCESSING

The above standards prescribe formal methods but give no advice as to which to use. To comply to these standards formal methods will have to be adopted by the control engineer. Some of the more mature formal methods which are applicable in the parallel processing domain are CSP, CCS, CSR, Petri nets and Temporal Logic, and their associated support tools, PAM, Concurrency Workbench and Design/CPN.

4.1. Coloured Petri nets

Coloured Petri nets [Jen–90] are an extension to Place/Transition nets and a refinement of Predicate/Transition nets. Petri nets express features of parallel processing systems in a simple, clear and graphical manner, and succinctly describe the behaviour of parallel systems. Coloured Petri nets allow tokens to represent data values from a multiset or *colour* set. Resultant nets are generally smaller than the equivalent Place/Transition net, because similar parts of the net may be folded into one, and the distinction between parts of the net can be made by the identity of the tokens. Tokens can now represent complex data objects, the notion of a colour set is analogous to data typing in programming languages. The example Petri net in figure 2 demonstrates the main features of coloured Petri nets (the example is an extract from some multipriority scheduling model). A colour set *Process* has been defined which is a tuple of priority and state for some simple scheduling system. Expressions on the input arcs to the two transitions specify which values from the multisets of input places may move down the arcs. The backquote in the expression `1`(pri, state)` is the *multiset operator*, in this case it states that just one instance of the tuple moves down the arc. *Transition guards* such as `[pri>0]` are Boolean expressions that guard the firing of a transition. A transition can only fire when the guard expression becomes true. Output arcs may contain *arc functions* which generate new token values. In this example, the function next() changes the value of a process's state. The modelling power of a Petri net is greatly increased with just these few simple features.

4.2. Temporal extensions for real–time analysis

Equally important to the design of a real-time systems is the analysis of its temporal behaviour. This is useful for two reasons. Firstly, it allows quantitative statements to be made about real–time properties of the system. Secondly, this *timewise* refinement of the design removes some of the non–determinism characteristic untimed system. As a design becomes more deterministic, it becomes easier to understand. Provided that the design incorporates a faithful representation of actual program instruction timings and time constraints, it would appear sensible to add as many timing details as possible to the design. This assumes a design environment that can tell when design changes affect these timing properties, and alert the designer when this happens. Welch [Wel-93] suggests that we can make no assumptions about the mechanisms of any multi–processing scheduler, and that relying on such properties to verify the absence of deadlock is implementation dependent and, therefore, unsafe. However, Suonio *et al* state that it is worthwhile to include implementation dependent information into the system model, because this stops the designer relying on implicit assumptions about the system, and all the required mechanisms are made explicit in some stage of the design [Suo–92].

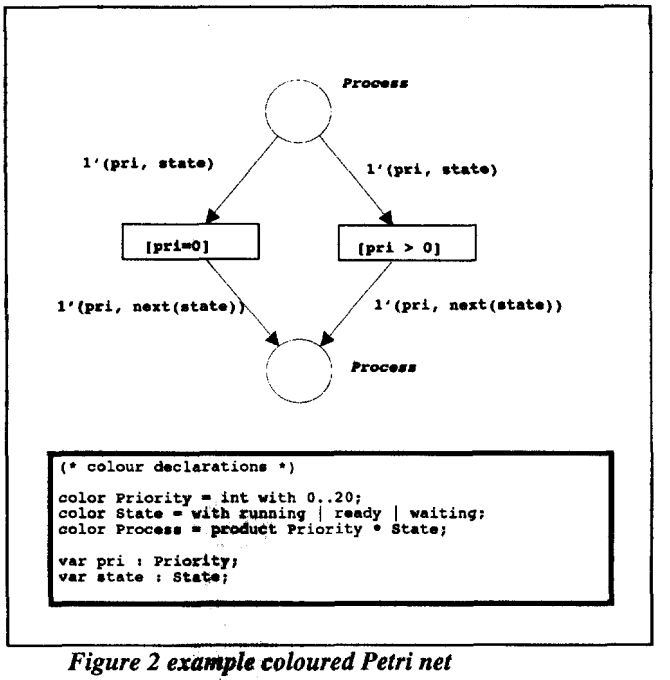

Figure 2 example coloured Petri net

4.3. Automated tools: The way forward

The potential for automation of some of these formalisms would reduce the requirement for the designer to be an expert in every detail of the method. This ensures the engineer can concentrate on the "control problem" involved. Automated tools which support some of the formalisms are presented. For example, Figure 3 shows an example using a Petri net design tool called Design/CPN. Such formal methods may be integrated into the design process by abstracting the mathematics behind a graphical veneer.

Software through Pictures (StP) is a CASE package developed by IDE. It provides an open environment to which user applications may easily be added. It allows an integrated systems approach which is guaranteed to provide consistency between tools. The tool is specifically designed to support structured methods such as Yourdon. StP and other traditional CASE tools suffer from not being able to express a formal model of the design and cannot therefore be used for mathematical analysis.

Thomas [Tho–93] suggests that if one cannot write down a formal description of the system, then one does not understand it. He goes on to suggest that the time and effort in training engineers to understand and use formal methods (in this case Z) is no more than for a structured method such as SSADM or Yourdon. This should give us hope for a more widespread takeup of formal methods and tools.

Design/CPN [Met-93] is a commercial CASE tool that fully supports coloured Petri nets. The designer builds a model through a graphical interface and the tool automatically checks the syntax of the diagram. In this way, the designer can construct models without requiring detailed knowledge of the underlying net theory. Simulation of the net is possible, and this interaction with the model can be thought of as an executable specification, or even a route to rapid prototyping. Many of the features of the tool can be accessed through a functional programming language called ML, allowing extra facilities to be added very simply. Design/CPN adds two features to coloured Petri nets; time and hierarchical decomposition. A duration can be specified for any transition which causes a delay before firing can occur. Any output tokens inherit a timestamp value which is the current value of a monotonically increasing global clock. This model of time is simple but intuitive. Hierarchy is supported by having substitution transitions which are an abstraction of a more detailed net specified in a sub-net below. This allows for decomposition of the

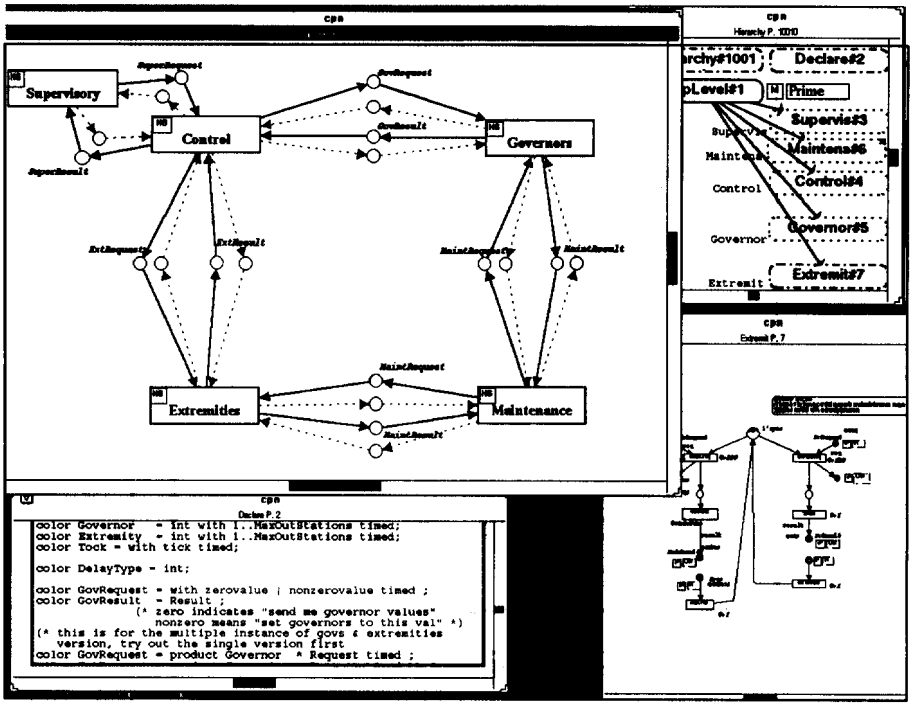

Figure 3. *A coloured Petri net tool.*

system in either a bottom-up or top-down manner. This is a major step forward in the acceptance of Petri nets for modelling purposes.

Petri nets, process algebra and temporal logic have close associations, and the fact that the three can be highly integrated suggests that a hierarchical Petri net model of a system is an ideal way of building understandable systems that are amenable to the rigourous design and analysis techniques demanded of safety-related systems.

5. CONCLUSIONS

Assuming parallel processing technology will continue to be used in high performance software controlled applications, no one particular standard addresses all of the issues involved. Present and future software safety standards will enforce the use of formal methods. As a consequence of this, control engineers must be competent in the use of formal reasoning. A broad perspective on the skills the control engineer must master was presented through a review of the maturer formalisms, and their associated problems and tools. The only way these techniques can realistically be applied is by the introduction of automated tools into a well structured design environment. Future research must be directed towards *push-button* formal methods. Formal methods are succinct, can have fairly simple semantics and can describe parallel behaviour, time and safety properties. They allow us to reason about the behaviour of system and more fully understand it.

Formal methods and structured methods (supported by StP) may be used together on the same design. There is not yet a strong base of software engineers experienced in formal methods, so currently it would appear more practical to take an StP design, identify certain specifically safety–critical parts of the system and model their behaviour in a more formal manner.

Methods required by the standards actually empower the designer with greater understanding of the system under study and provide a platform for rigorous analysis of the correctness of the design. Formal methods that are graphical and intuitive, such as Petri nets, can be used proficiently be engineers with a minimum of training.

Parallel processing is sufficiently mature for developing safety-related control applications, but designs will need to be integrated with automated formal analysis tools.

6. REFERENCES

[Ame–93] Amerongen, J. et al, "The Twente Approach to System Level Embedded Controller Design", Proceedings of the 5th WoTUG/Japan meeting, pp. 22–35, 1993.

[Aus–93] Austin, S., Parkin, G., "Formal Methods: A survey", National Physical Laboratory, March 1993.

[Bir-93] Birkinshaw, C.I., Croll, P.R., Marriott, D.G., Nixon, P.A., "Engineering Safety-Related Parallel Systems", Software Engineering for Parallel Systems Workshop, WTC–93, Aachen, Germany, September 1993.

[Ben–93] Bennet, P., "Safety Critical Systems: So What...?", IMechE & IMeasC Symposium on Systems and Safety, May 1993.

[Cro-91] Croll, P.R., Nixon, P.A., "Developing safety-critical software within a CASE environment", IEE colloquium on 'Computer Aided Software Engineering tools for real-time control', No. 1991/087, 24 April, 1991.

[Irw-92] Irwin, G.W., Fleming, P.J., "Real–time control applications of transputers", In Transputer Applications: progress and prospects, Ed. M.R. Jane, R.J. Fawcett, T.P. Mawby, IOS Press, 1992.

[Jen–90] Jensen, K., "Coloured Petri Nets: A high level language for system design and analysis", Lecture Notes in Computer Science, Ed. G. Rozenberg, Advances in Petri Nets, Springer-Verlag, 1990.

[Met–93] Meta Software Corporation, "Design/CPN Reference Manual", 125 Cambridge Park Drive, Massachusetts, USA, 1993.

[Red–93] Redmill, F., Anderson, T. (Eds.), "Safety–critical systems: Current issues, techniques and standards", Chapman & Hall, 1993.

[Suo-93] Kurki–Suonio, R., Systä, K. & Vain, J., "Scheduling in Real-Time Models", Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 571, pp 327-339, 1992.

[Tho–93] Thomas, M., "The industrial use of formal methods", Microprocessors and Microsystems, Vol. 17, No. 1, January/February 1993.

[Wel-90] Welch, P.H., "Parallel algorithms and safety-critical standards", Presented at Benelux meeting on systems and control, Eindhoven Holland, 14-16 March, 1990.

[Wel–93] Welch, P.H., Justo, G. & Willcock, C., "High–Level Paradigms for Deadlock–Free High–Performance Systems", Proceedings of WTC 93, Aachen, Germany, September 1993.