Measuring concurrency in CCS

Vashti C. Galpin

Department of Computer Science, University of the Witwatersrand

vashti@concave.cs.wits.ac.za

October 19, 1999

Abstract

This paper will report on research which investigates the application of Charron-Bost's measure of concurrency m to the Calculus of Communicating Systems (CCS), a formalism for expressing concurrency. The purpose of a measure of concurrency is to calculate a numerical value that describes the amount of concurrency occurring in a computation or algorithm. The aim of the research was twofold: first to evaluate the measure m in terms of criteria gathered from the literature, and second to determine the feasibility of measuring concurrency in CCS and to provide a new tool for understanding concurrency using CCS. The approach taken in the research was to identify the differences between the message-passing formalism in which the measure m is defined, and CCS; and to modify the message-passing formalism to enable the mapping of CCS agents to it. A software tool, the Concurrency Measurement Tool, was developed to permit experimentation with chosen CCS agents and hence to allow the evaluation of m. The criteria used for evaluation are the intuitive understanding of the measure, being well-behaved for small examples, compatibility with operators, usability and applicability, expense of computation, ability to calculate the measure for a specific event and stability with respect to granularity. The experimentation showed that the measure m, although intuitively appealing, is defined by an algebraic expression that is ill-behaved, is not compatible with operators and is expensive to compute. A new measure is defined and it is shown that it matches the evaluation criteria better than m, although it is still not ideal. The research has demonstrated that it is feasible to measure concurrency in CCS and that a methodology has been developed for evaluating concurrency measures.

1 Introduction

As Hoare has noted 'Concurrency remains one of the major challenges facing Computer Science, both in theory and in practice' [19, Foreword]. For this reason, the investigation of issues that relate to concurrency is important, as such an investigation can further our understanding of concurrency.

This paper reports on the research that was presented for the degree of Master of Science at the University of the Witwatersrand [13]. In this research, I evaluated a measure of concurrency, namely Charron-Bost's measure m [6] by applying it to an algebraic calculus of processes, namely Milner's CCS (Calculus of Communicating Systems) [19] and in the course of this evaluation I developed a methodology for evaluating concurrency measures.

Algebraic calculi of processes provide an approach to modelling communication and concurrency by describing the behaviour of concurrent communicating systems. CCS is an example of this type of calculus. It provides for the definition of agents using a small number of operators and for the definition of the behaviour of these agents via an operational semantics, hence allowing for the description and specification of concurrent systems, from which an understanding of the behaviour of such systems and the comparison of

different systems can be gained. Any new tools developed for use with CCS will further the understanding of concurrent systems.

In the literature, there are a number of approaches to analysing the behaviour and performance of concurrent systems. Lamport and Lynch [18] note that there are two basic complexity measures for distributed algorithms: time complexity which measures the amount of time taken by message passing in a particular algorithm; and message complexity which is based on the number of messages or the total number of bits transmitted during a computation. Parallel programs also exhibit concurrency and their performance can be measured by speedup; the ratio of time taken to execute the program on one processor and the time taken on n processors [20].

However, as Charron-Bost has noted [6], none of these measures assess the structure of the computation. She gives the example of a multibroadcast problem with two solutions—the solution with the lower number of messages works in an essentially less concurrent manner and is less resistant to failure than the solution with the higher number of messages. Recently in the literature, the notion of a concurrency measure has been proposed by a number of authors [1, 2, 3, 4, 5, 6, 10, 12, 14, 15, 16, 21]. This notion is based on the concept of a quantitative measure of the amount of concurrency in a computation or algorithm, usually defined on the interval $[0, 1]$ or $[0, \infty)$.

Although there have been a number of different measures proposed, there has been a lack of experimental results for these measures, and the majority require further evaluation. Generally, it appears that researchers present measures but little further research is done to evaluate the usefulness of the defined measures in more practical situations. One aspect of this research is to remedy this situation by first investigating one specific measure, and second by proposing a general approach for evaluating measures.

To evaluate measures of concurrency, it is necessary to determine which features or characteristics are generally desirable. Dispersed throughout the literature are a number of different criteria for evaluating measures of concurrency. These different approaches are drawn together in this research and will be used to evaluate the measure of concurrency under investigated.

The structure of the paper is as follows: the first two sections are overviews of existing work—in Section 2, CCS will be briefly discussed, and in Section 3, the measure m will be presented. In Section 4, the application of the measure m to CCS will be discussed. Then in Section 5, the program that was written to calculate the measure for CCS agents will be described. In Section 6, the criteria for the evaluation of a measure of concurrency will be described and in Section 7, the experimental results of the research will be presented. Finally I will discuss the conclusions from the research and mention some issues that remain for further research.

2 CCS

For reasons of space, I will not discuss CCS in much detail and the reader is referred to [19] for further details. CCS provides the following:

- A syntax of operators over a set of events, Act.
  - Nil $\mathbf{0}$, Prefix $a.$, Summation $+$, Composition $|$, Restriction $\backslash a$, Relabelling $[f]$, Constant (or Recursion).
- Operational semantics, specifying the behaviour of operators which results in a transition system, namely the transition rules:
  - **Act**, **Sum**, **Com**, **Res**, **Rel**, **Con**.
- Semantics given by an equivalence, for example strong equivalence and observation equivalence.

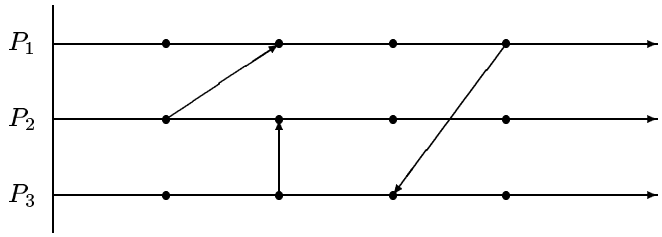

Figure 1: A distributed computation represented as a space-time diagram

In this research, the approach taken in applying the measure of concurrency to CCS will involve mapping the syntax and indirectly the operational semantics to a formal model of a distributed system. However, it will not involve equivalence semantics.

3 The message-passing formalism and Charron-Bost's measure m

The measure m is defined in the model of message-passing first defined by Lamport [17], which I shall refer to as the message-passing formalism. This formalism defines a formal model of a distributed system and allows for the definition of a partial ordering on the events in the distributed system, called the 'happened before' relation. This framework has been formalised by Charron-Bost [5] as follows:

A distributed system consists of a finite set of sequential processes $\{P_1, \ldots, P_n\}$ where each P_i is a set of sequences of events and is prefix closed. Events fall into three classes: an internal event, the sending of a message to another process, or the receipt of a message from another process.

For each i, let C_i be one of the sequences defining P_i. The relation $\prec$ is defined on the set of events obtained from the computation $C = C_1 \cup \cdots \cup C_n$ as the smallest transitive relation satisfying (1) if a and b occur in the same process and a comes before b, then $a \prec b$, (2) if a is a sending of a message m, and b is the receipt of m, then $a \prec b$. This relation, known as the 'happened before' relation forms a partial order on the set of events and captures the causal relationship between events.

$\preceq$ is defined as: $a \preceq b$ iff $a \prec b$ or $a = b$. Events a and b are concurrent (a co b) if $\neg(a \preceq b)$ and $\neg(b \preceq a)$. Concurrent events are those that are causally independent of each other.

The happened-before relationship can be captured in the space-time diagrams introduced by Lamport [17]. In Figure 1, the horizontal lines represent processes, the time axis runs from left to right, and send and receive events are joined by a directed line.

The measure m is based on counting the number of consistent cuts in a computation. The concept behind this is that the more processes wait, the less concurrency the computation exhibits. Since the ability of the computation to be cut consistently is related to its tolerance to stopping, a computation's concurrency can be measured in terms of its number of consistent cuts.

A cut of a distributed computation can be defined as follows: Consider the union where each a_i is an element of C_i

$$C' = \bigcup_{i \in \{1,\ldots,n\}} \{x \in C_i \mid x \preceq a_i\}.$$

This is called a cut and it may contain the receipt of a message but not the sending of that message, therefore it may exhibit inconsistency.

A consistent cut can be viewed as a global state of the computation where causality is not violated [11], hence the consistent cut cannot contain the receipt of a message without its sending. A consistent cut of a distributed computation C is defined as a cut C' that is left closed by the causality relation. This means that for any $a, b \in C$, if $b \in C'$ and $a \preceq b \Longrightarrow a \in C'$.

The measure m is defined as

$$m = \frac{\mu - \mu^s}{\mu^c - \mu^s}$$

where μ is the number of cuts in the computation under consideration, μ^s is the number of cuts in a totally sequential version of the computation (where no events can occur concurrently) and μ^c is the number of cuts in a totally concurrent version of the computation (where there is no communication). The measure takes values in the interval $[0, 1]$ with 1 for a concurrent computation and 0 for a sequential computation.

The values of μ^s and μ^c can be determined analytically. Assuming there are q_i events in each process then

$$\mu^s = 1 + q_1 + \cdots + q_n,$$

$$\mu^c = (1 + q_1) \cdots (1 + q_n).$$

Vector logical clocks [11] can be defined on distributed computations and Charron-Bost has proved that vector logical clocks can be used to check cuts for consistency.

Raynal [21] describes m as a good characterisation of the degree of concurrency, but he notes that its computation is not feasible. Fidge [10] notes that the calculation of μ using vector clocks requires $\sum_{1 \leq i \leq n} nq_i$ integers to be stored and that m can only be calculated after the computation has terminated. Charron-Bost [7] has shown that m does not give the expected results when used with the operators concatenation and fusion which are operators that are defined within the message passing formalism. Finally, she notes that there may be other operators that are better adapted for use with m, but she is not aware of any that are easily studied from the point of view of combinatorics, so part of this research will evaluate m using the operators found in CCS. The measure has not been evaluated in terms of applicability to real examples.

Hence it appears that there is conflicting evidence regarding m. It has been described as a good measure of concurrency, although it is computationally expensive and is not compatible with operators; however, further work is required in these areas.

4 Applying the measure to CCS

To enable the application of the measure to CCS, it was necessary to determine the differences between the message-passing formalism and CCS. The major differences can be detailed as follows:

1. Charron-Bost's formalism assumes that there are a fixed number of processes that all start at the same time, and that no new processes are created during the computation. In CCS, certain agents can be interpreted as involving process creation, for example $a.(b.\mathbf{0} \mid c.\mathbf{0})$, starts as a process that performs a, and then 'branches' into two processes, one performing performs b and one c.

2. Charron-Bost's formalism assumes asynchronous communication. In CCS, communication occurs between a port a and its complement $\overline{a}$ and this communication is synchronous.

3. There is a difference in expressive power with respect to computations. The message-passing formalism represents one computation of an algorithm whereas a CCS agent can describe an entire algorithm.

The first two differences are resolved by redefining the message-passing formalism to allow for synchronous communication and for creation of processes. This redefinition results in a new relation on $\{P_1, \ldots, P_n\}$, namely $\mapsto$ and a new relation $\leftrightarrow$ on events. The relation $\mapsto$ describes causality between processes, for example, $P_i \mapsto P_j$ means that process P_j occurs after and is created by process P_i. The relation $\leftrightarrow$ describes pairs of communication events, for example $a \leftrightarrow b$ means that synchronous communication occurs between a and b.

The relation $\preceq$ is defined on events and describes the causal relationship between events, as previously; however, it now takes into account synchronous communication and process creation, and in the partial ordering of events, synchronous communication events are equated—they are viewed as a single event.

It is also necessary to investigate the definition of cuts. This definition requires slight modification since Charron-Bost's original definition specifies that each process contributes an event to the cut which excludes the definition of some cuts, for example the empty cut. The definition is modified as follows: let $C = C_1 \cup \ldots \cup C_n$ be a computation, let $N = \{1, \ldots, n\}$, and let $E_{C'} \subseteq N$. $E_{C'}$ will contain the indices of the processes that do not contribute any events to the cut C'. Then for each $i \in N \setminus E_{C'}$, choose an $a_i \in C_i$, then

$$C' = \bigcup_{i \in N \setminus E_{C'}} \{x \in C_i \mid x \preceq a_i\}$$

is a cut of the computation. A cut C' is a consistent cut if it is left closed by $\preceq$, namely, if $a, b \in C$ such that $b \in C'$ and if $a \preceq b$ then $a \in C'$.

Vector clocks can also be defined in the redefined formalism and it has been proved that vector clocks can be used to determine if a cut is consistent, in a similar fashion to Charron-Bost's result[1].

It can be seen that intuitively the basic idea of the measure m holds in this redefined formalism, namely that consistent cuts indicate how resistant to stopping the computation is; however, there are new values for μ^s and μ^c.

$$\mu^s = 1 + q_1 + \cdots + q_n - |\leftrightarrow|$$

The value of μ^s is reduced by the number of communication events since communication events are equated.

$$\mu^c = \prod_{P_i \in \mathcal{S}} \text{cuts}(P_i) \qquad \text{where} \qquad \text{cuts}(P_i) = q_i + \prod_{\{P_j \mid P_i \mapsto P_j\}} \text{cuts}(P_j)$$

This more complex definition for μ^c occurs because although there is no communication in a totally concurrent computation, the causality between processes must now be taken into account.

Finally, the difference in expressive power was investigated. In the message-passing formalism, each computation represents a specific execution of an algorithm, whereas in CCS, an agent represents an algorithm which can generate many computations. Under certain conditions, it is easy to identify separate computations, since they only occur as a result of the Summation operator. Hence for any CCS agent which meets these conditions, many computations may be identified, and the concurrency for each will be measured. The measure for the original agent will then be defined as an ordered pair, consisting of the maximum and minimum values found across all computations.

The outline of the translation procedure is given below. This procedure assumes that the different computations have been identified.

- CCS actions will be mapped to events in the message-passing formalism, and actions and their complements will be used to identify communication event pairs. When a communication event occurs, the two actions are replaced by a single τ action. For example, consider the following agent

[1] The statement and proof of this result is given in [13].

$(a.\mathbf{0} \mid \overline{a}.\mathbf{0})\backslash a$. The communication event that occurs between a and $\overline{a}$ will be represented by the single action τ.

- An agent of the form $a_1.\ldots.a_n.\mathcal{Q}$ where $\mathcal{Q}$ is a CCS agent that does not start with a prefixed action, will be identified as a process. So, for example, the agent $a.b.(c.d.\mathbf{0} \mid e.f.\mathbf{0})$ will be decomposed into three processes—$P_1 = ab$, $P_2 = cd$ and $P_3 = ef$, with $P_1 \mapsto P_2$ and $P_1 \mapsto P_3$.

Another issue of importance is to determine what subset of CCS can be measured. Note first of all that the message-passing formalism assumes that processes eventually terminate, therefore only finite agents will be used. Milner [19, p. 160] defines finite agents as follows: '[a]n agent expression is finite if it contains only finite Summations and no Constants (or Recursions).' Hence the subset of CCS that will be dealt with is contained in the subset of agents defined by $\mathbf{0}$, Prefix, (finite) Summation, Composition, Restriction and Relabelling. As noted above, there are some additional conditions on agents and it can be shown that the subset of CCS that can be handled within these conditions includes finite confluent agents. Milner [19] notes that confluence is a desirable property for ensuring well-behaved specifications. This would indicate that although it would be preferable to capture the whole of CCS, the subset that has been dealt with is, in fact, significant.

The algorithm that will be used is a static algorithm to translate a CCS agent into the redefined formalism. It is static in the sense that the decomposition of an agent into processes is based only on the syntax of the agent. The input is a CCS agent and the output is a set of processes $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, consisting of sequences of events, a relation that describes the causal links between processes $\mapsto$, and a relation that describes the communication pairs $\leftrightarrow$. An example of a translated CCS agent is given in Figure 2. In the following, *agent* will be used to indicate the CCS agent and i, j, k will indicate process numbers.

translate(*agent* , i)
    if *agent* has the form $a.E$ then
        add a to P_i
        translate(E, i)
    if *agent* has the form $E \mid F$ then
        get new process numbers j and k
        create empty processes P_j and P_k
        add (P_i, P_j) and (P_i, P_k) to $\mapsto$
        translate(E, j)
        translate(F, k)
    if *agent* has the form $\mathbf{0}$ then
        do nothing
    if *agent* has the form $E\backslash a$
        add the action a to the restriction information about E
        translate(E, i)

identify communication pairs and create $\leftrightarrow$

5 Concurrency Measurement Tool

As part of this research, a program was written that calculated the measure of concurrency for CCS agents in the subset described in the previous section. The program performed the translation described above and then calculated the measure for the resultant space-time diagram. The approach taken was to calculate

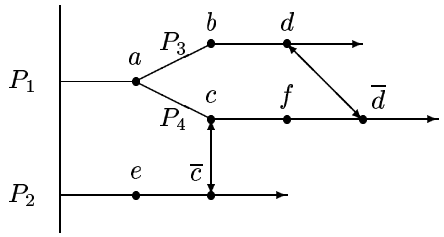

Figure 2: The translation of the agent $(a.(b.d.\mathbf{0} \mid c.f.\overline{d}.\mathbf{0}) \mid e.\overline{c}.\mathbf{0}) \setminus c \setminus d$.

μ by generating each cut and then using the vector clocks to check the cut for consistency. This was not an efficient algorithm and it has been shown to be $O(q^n/n^{n-2})$ where q is the total number of events in the computation and n is the number of processes. This analysis was confirmed by experimental timings obtained from running the program.

Other approaches to this algorithm are also problematic. Charron-Bost [5, 7] has derived an algorithm based on her result that relates the number of consistent cuts to the number of antichains in the partial order. She notes that the algorithm is efficient only when there are few 2-antichains in the computation under consideration, namely when there is little concurrency in the computation. I have investigated this algorithm further and found it to be $O((nq)^{n+1})$. Kim et al [15] present an approach whereby they divide computations into concurrency blocks, and count the number of antichains in each block. This results in an approximation to m that is faster to compute; namely $O((nq)^n/b^{n-1})$, where b is the number of concurrency blocks; however no work has yet been done on the accuracy of the approximation.

6 Criteria for evaluation

The evaluation criteria that have been gathered from the literature are:

- Intuitive understanding of the measure [5]
- Being well behaved for small examples [5, 6]
- Compatibility with operators on computations [6, 8, 10]
- Usability and applicability to actual algorithms [6, 16]
- Ability to calculate measure for a specific event [10, 21]
- Expense of computation in terms of both time and space [10, 14, 21]
- Stability with respect to granularity [7]

These criteria will not be discussed in detail for reasons of space[2]. However it should be noted that one important criterion is that of compatibility with operators. Operators on concurrent computations allow computations to be 'joined' in some sense. Examples of such operators are parallel concatenation and sequential concatenation. When two computations are combined by an operator, it would be desirable that the value of the measure of concurrency for the combined computations can be explained in terms of the type of operator. For example, when computations are combined in parallel, it is reasonable to

[2] A more detailed explanation of these criteria can be found in [13].

expect the value of the measure to increase as the computation has in some sense become more parallel. Compatibility with operators is an important criterion when building systems from components. If the effect of an operator can be explained, then the effect on the amount of concurrency when exchanging components can be determined.

7 Experimentation and results

A number of experiments were performed using the Concurrency Measurement Tool. The experiments performed can be grouped into two categories— experiments on simple CCS agents and experiments on CCS implementations of two solutions to the Dining Philosophers problem.

The first category of experiments showed that the measure appeared to be well-behaved for simple examples. However, when the various operators were used to build new systems from simple examples, anomalous results occurred. When the Prefix operator (which can be seen as a sequential operator) was used, the measured amount of concurrency did not change, although it was expected to decrease. Likewise, when the Composition operator (which is a parallel operator) was used, the amount of concurrency measured decreased which also differs from the expectation that it would increase.

In the Dining Philosophers experiments, the resulting values of the measure were extremely small (of the order of 10^{-6}) and did not appear to distinguish between different agents. For example, the value for the agent for two philosophers was 5.78×10^{-3} and for four philosophers it was 9.79×10^{-5} which would appear counter-intuitive since it would be expected that there would be more concurrency in the case where three out of four philosophers can eat simultaneously as opposed to the case where only one out of two can eat at a specific time. Therefore, this shows that there are algorithms for which the measure does not behave well.

After an investigation of the values of μ, μ^s and μ^c it was determined that μ^c dominates the value of m. Hence, the values obtained in the experiment are small because μ^c occurs in the denominator. From this, it can be argued that the algebraic expression for m is ill behaved.

In an attempt to solve these problems, a new measure was defined

$$m_{\text{new}} = \frac{\log \mu - \log \mu^s}{\log \mu^c - \log \mu^s}.$$

This form was chosen because taking logarithms reduces the size of the values, and should thus prevent μ^c from dominating the measure.

On performing the same experiments with m_{new}, it was found that $m_{\text{new}}$was better behaved. It was more compatible with operators although it was still possible to find examples where the use of the Composition operator could cause a reduction in measured concurrency. It also returned larger values in the experiments on the Dining Philosopher agents, and it was found that the case of two philosophers had less measured concurrency than the case of four as was originally expected. Note also that $m_{\text{new}}$is as expensive to calculate as m. Therefore the measure m_{new} is better than m although it does not fully solve all problems associated with m.

In the course of this research it has been shown repeatedly by the fact that the evaluation of the measure m was possible that the concept of measuring concurrency in CCS is a workable one. Although the measures have been only defined for a subset of CCS, this subset does include confluent agents.

8 Conclusion

This research has showed that concurrency can be measured in CCS and that it can be used to evaluate concurrency measures. Hence, the approach taken here to evaluate m can be used to evaluate other

measures. This will provide both for evaluation of existing measures and the development of new measures; and for the further development of a new tool for the theoretical investigation of concurrency using CCS.

A number of issues for further research have come to light as a result of this research.

- Since m_{new}does not solve all the problems associated with m, an area of research would be to develop a better measure based on consistent cuts.
- Research can be done to find a better algorithm (if possible) to count the number of consistent cuts in a computation.
- The existing framework can now be used to evaluate other measures of concurrency that have been defined in the message-passing formalism.

When authors present measures of concurrency in literature, generally they do not present evaluations of these measures. For a measure to be a useful tool, it is essential that it is evaluated with respect to relevant criteria. The research has drawn together a number of criteria and used them in conjunction with the Concurrency Measurement Tool to evaluate m. This approach can also be applied to other measures defined in the message passing formalism.

Therefore, an important outcome of this work has been to produce a methodology for evaluating measures of concurrency. This approach currently can be used for measures defined in the message-passing formalism; however, it may be possible to use CCS with measures defined in other formalisms. It has been shown to be a useful methodology by the fact that it facilitated the discovery that a specific measure was ill behaved and allowed for the definition of a measure using the same basis that fitted better with the evaluation criteria.

Acknowledgements I wish to thank my supervisor Scott Hazelhurst for his guidance, insight, time and encouragement. This research was supported by the Foundation for Research Development.

References

[1] D. Arques, J. Françon, M. Guichet, and P. Guichet. Comparison of algorithms controlling concurrent access to a database: a combinatorial approach. In L. Kott (ed), *Proceedings of 13th ICALP*, Lecture Notes in Computer Science **226**, 11–20. Springer-Verlag, 1986.

[2] N. Bambos and J. Walrand. On stability and performance of parallel processing systems. *Journal of the ACM*, **38**(2), 429–452, April 1991.

[3] V. Barbosa. Blocking versus nonblocking interprocess communication: a note on the effect on concurrency. *Information Processing Letters*, **36**, 171–175, 1990.

[4] J. Beauquier, B. Bérard, and L. Thimonier. On a concurrency measure. Technical report, L.R.I., Univ. Paris-Sud, Orsay, 1986.

[5] B. Charron-Bost. Combinatorics and geometry of consistent cuts: application to concurrency theory. In J.-C. Bermond and M. Raynal (eds), *Distributed Algorithms*, Lecture Notes in Computer Science **392**, 45–56. Springer-Verlag, 1989.

[6] B. Charron-Bost. Measure of parallelism of distributed computations. In B. Monien and R. Cori (eds), *STACS '89*, Lecture Notes in Computer Science **349**, 434–445. Springer-Verlag, 1989.

[7] B. Charron-Bost. *Mesures de la concurrence et du parallélisme des calculs répartis.* PhD thesis, Informatique, Universite Paris VII, 1989.

[8] B. Charron-Bost. Coupling coefficients of a distributed execution. Technical Report 92.07, Laboratoire Informatique Théorique et Programmation, Institut Blaise Pascal, Paris, 1992.

[9] E. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, **1**, 115–138, 1971.

[10] C. Fidge. A simple run-time concurrency measure. In T. Bossomaier, T. Hintz, and J. Hulskamp (eds), *The Transputer in Australasia (ATOUG-3)*, 92–101. IOS Press, 1990.

[11] C. Fidge. Logical time in distributed computing systems. *Computer*, **24**(8), 28–33, August 1991.

[12] J. Françon. Une approche quantitative de l'exclusion mutuelle (A quantitative approach to mutual exclusion). *Theoretical Informatics and Applications*, **20**(3), 275–289, 1986.

[13] V. Galpin. Measuring concurrency in CCS. Master's thesis, Department of Computer Science, University of the Witwatersrand, 1993.

[14] D. Geniet and L. Thimonier. Using generating functions to compute concurrency. In J. Csirik, J. Demetrovics, and F. Gécseg (eds), *FCT '89*, Lecture Notes in Computer Science **390**, 185–196. Springer-Verlag, 1989.

[15] M. Kim, S. Chanson, and S. Vuong. Concurrency model and its application to formal protocol specifications. In *IEEE Infocom '93*, 1993. To appear.

[16] M. Kumar. Measuring parallelism in computation-intensive scientific/engineering applications. *IEEE Transactions on Computers*, **37**(9), 1088–1098, September 1988.

[17] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, **21**(7), 558–565, July 1978.

[18] L. Lamport and N. Lynch. Distributed computing: models and methods. In J. van Leeuwen (ed), *Handbook of Theoretical Computer Science*, Volume B, 1158–1199. Elsevier Science Publishers, 1990.

[19] R. Milner. *Communication and concurrency.* Prentice Hall, 1989.

[20] M. Quinn. *Designing efficient algorithms for parallel computers.* McGraw-Hill, 1987.

[21] M. Raynal, M. Mizuno, and M. Neilsen. Synchronisation and concurrency measures for distributed computations. Technical Report 1539, INRIA, October 1991. To appear in IEEE International Conference on Distributed Computer Systems 1992.