Parallel processing in large-scale simulation: motivation and methods

This article discusses the use of parallel computers in the context of modern control engineering practices. It outlines potential advantages, the role of the control engineer, and suggests methodologies, with particular reference to the problem of simulating large-scale systems. Many of the ideas introduced are general enough to extend to a much wider class of problems.

by Anthony W. Burton

Lucas Automotive Ltd.

Foundations for the modern systematic approach to control engineering were laid by James Maxwell, who was probably the first person to fully understand the connection between a physical system and its mathematical model. An early stage of his now well-established design cycle is designated *analysis* and *modelling*. This process captures the essence of the problem, develops insight, and indicates what form of controller synthesis technique is likely to be most appropriate in meeting design specifications.

Since the earliest days of electronic computation, much of the analysis and modelling work of control engineers has been conducted via simulation. Nowadays, this setting up of a (usually digital) computational homology of the dynamics of a physical system is an accepted integral part of control engineering practice. This practice, however, has all too often been associated with immoderate or (in some cases) excessive demands on processing time.

The problem arises that, as physical systems evolve and become more complicated, the dimensionality (or number) of the underlying dynamical equations necessary to describe them can increase dramatically. This in turn can lead to immense computational burdens when attempting any simulations (which require a numerical approximation to the solution of these equations at regular time intervals). Often this may cause difficulties in meeting real-time constraints (as may exist in an operator-training simulator), and the effects may become so acute that simulations become impractical to conduct because execution times are so long.

One of the most exciting technological developments which promise to alleviate compute-bound problems, such as the simulation of large-scale systems (LSS), is that of so-called 'concurrent' or 'parallel' processing. This article motivates the use of concurrent computing as an alternative to the traditional (von Neumann) approach by first highlighting those fundamental characteristics which distinguish the two. Arguments are then put forward which indicate why the intimate relationship between state-of-the-art electronic computing and control engineering should be extended to the parallel regime. Ultimately, the main thrust of the work is to suggest an integrated approach to exploiting the new processing opportunities afforded by parallel architectures. This is done mainly with reference to LSS simulation as a 'theme' problem, where it is implicitly assumed that plant dynamics are modelled by a system of first-order ordinary differential equations (ODEs). It should be noted, however, that many of the concepts considered are general enough to logically extend to a much wider class of problems.

Motivation

Why go parallel? In order to justify any further consideration of applying parallel processing techniques, there is a need to fully appreciate the limitations imposed by the alternative option of traditional computer technology. This alternative is also an important frame of reference on which many parallel processing concepts hinge.

Perhaps one of the first things to note about parallel processing is that it is not a new concept:[1]

'Those responsible for the designs of the very first electronic computers with internally stored (and hence machine modifiable) programs, namely von Neumann and Turing, were well aware of the significance of parallelism both in performing elementary operations, and in overlapping them with other operations such as input and output.

'... the extent of concurrency in actual early machines was limited more by engineering considerations (of the day), *than by the aspirations of some of the designers.'*

A second myth that should be dispelled at an early stage is that the

performance gains historically associated with successive generations of digital computer are directly attributable to advances in constitutive technology (component integration/fabrication methods). In fact, Hockney and Jesshope[2] show that, in the period from 1950 to 1975, the basic speed of components (as measured by the inverse of the gate delay time), has increased by a factor of about 10^3. In the same period, the performance of computers (as measured in MIPS) has increased by a factor of about 10^5. 'This additional speed has been made possible by architectural improvements, principally the introduction of parallelism'[2].

Given these considerations, the obvious question arising is where exactly does the parallelism in a conventional digital computer architecture lay, and does this mean that *all* computers are in fact parallel processors... or not? In order to answer these questions, consider the well-known universal model of the digital computer attributed to von Neumann (Fig. 1). The universal computer concept encompasses some fundamental elements associated with the way in which information processing occurs. These include:

- digital representation of data held in memory store
- a coded program of algorithms also held in memory store
- a single arithmetic logic unit (ALU) to execute the basic operations of arithmetic and logic
- a single control unit which decodes program instructions and monitors their execution on the ALU
- support of data input and output (I/O) functions.

Techniques exist which exploit parallelism at virtually every stage of the above, without recourse to a new conceptual model of the processes involved. One of the most common of these is the management of the transfer of data while the processor unit continues to compute. This is known as concurrent I/O. Another common feature is the use of bit-parallel arithmetic.

At the 'instruction-fetch' level, so-called 'look ahead' control units have been developed. The term look ahead derives from a class of schemes in which programs for the processor are specified in a conventional serial manner, but the control unit can look ahead during run time and manage instruction executions out of sequence, provided that no logical inconsistencies arise as a result.

The main motivation for look-ahead is in support of another aspect of parallelism, namely pipelining.

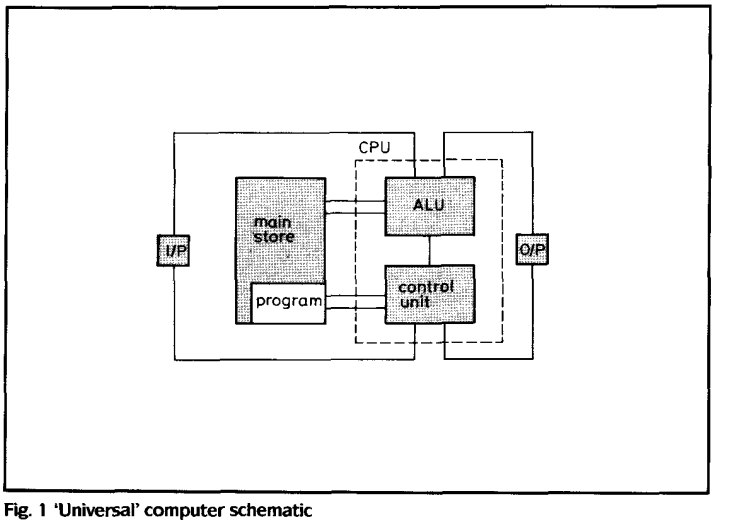

Fig. 1 'Universal' computer schematic

Pipelining relies on the fact that many computational processes require a number of clock cycles to complete if executed strictly sequentially. Provision of separate (but linked) functional units for each phase of the process is characteristic of pipelining. The resulting overlapping executions can increase throughput.

Memory access times have also been reduced by the introduction of parallellism. Fast cache systems may buffer main memory which itself may be interleaved in a pipelining fashion.

Overall, the conclusion must be that the definition of a concurrent digital computer must be made in the context that *all* computers have some degree of parallelism. The definition that will be adopted here is: A digital computer for which concurrent processing can occur, at a level not lower than that of an elemental arithmetic or logical operation.

There are two important corollaries of what has been discussed so far: first, the adopted definition of parallel processing precludes all of the previously mentioned parallel techniques; secondly, it is apparent that parallel processing is not a completely radical departure from conventional computing, but can be viewed as a logical extension to numerous other levels of concurrency that have been shown to exist already. Just as these other parallel methods have alleviated bottlenecks in memory access etc., so can parallel processing alleviate compute-bound problems such as LSS simulation.

Technical gap

Literally hundreds of parallel processing architectures have been proposed and many built. Often these novel architectures require changes in von Neumann's 'universal' model to describe them. These two facts combined often cloud the understanding of concurrent processing.

The parallel environment can appear even more complicated following the realisation that the vast majority of architectures are 'polyparallel'. In other words, most exploit computational parallelism at more than one level, and may (theoretically at least) employ any combination of hardware, operating system, compiler, or programmer in so doing. Fortunately for the control engineer, a large proportion of the complexity of computational polyparallelism is hidden by its 'implicit' representation to the hardware.

An implicit representation of parallelism implies that the mechanism by which concurrent processing is achieved is effectively hidden from the user. An explicit representation, on the other hand, requires that the concurrent processes are identified explicitly by the programmer. Obviously, the implicit approach is preferable in all cases since it provides a simpler user interface which ultimately supports improved programming productivity. Unfortunately, a 'technical gap' in the application of implicit methods occurs according to the average size and complexity (granularity) of the individual concurrent tasks.

At one extreme, if the tasks to be run in parallel are complete jobs or programs (coarse grained), it is the operating system that has the task of allocating resources and scheduling parallel executions. At the instruction execution and bit level, the characteristics of any compiler used are of paramount importance. Within

the remaining concurrency levels (medium granularity), central issues include the structure selection and scheduling of the applications tasks, as well as the choice of numerical algorithms.

Currently, these are issues that must be addressed within a technical gap (i.e. in an area for which there are few readily available automatic support tools). As a result it appears that the control engineer or systems analyst should be poised to explicitly make many of the crucial design decisions related to application-specific parallel processing. It is the examination and understanding of these explicit methods that is the objective of the remainder of this article.

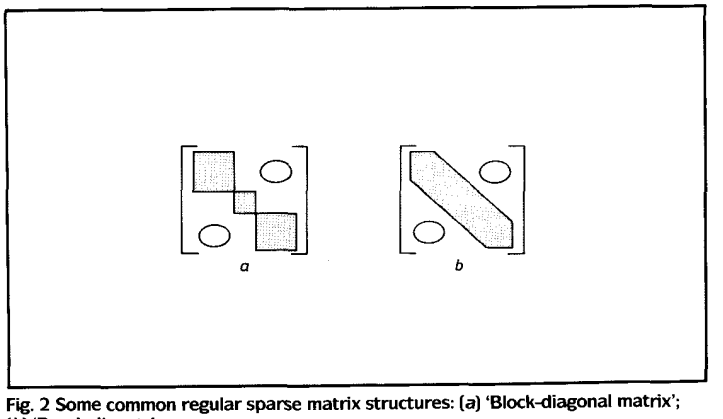

Fig. 2 Some common regular sparse matrix structures: (a) 'Block-diagonal matrix'; (b) 'Banded' matrix

Methods

Available methods for explicit concurrency exploitation may be classified according to whether they are applied *before*, *during*, or *after* discretisation (a process that must occur in order for the numerical solution of ODEs to be generated on digital computers). Consider each case in turn:

Pre-discretisation

The two traditional tools for dealing with the 'curse of dimensionality' are aggregation and matrix partitioning. Essentially, aggregation methods substitute a high-order system representation (mathematical model) by one of much smaller size, but which retains some poignant features of the physical system which are adequate for a particular problem.

Aggregation techniques have their roots in the socio-economic sciences where 'fuzzy' models are adequate (or only possible), and have been used so far rather more in such fields than in the area of technology where the intention, generally speaking, is to recreate the time behaviour of a system as faithfully as is practically viable.

Matrix partitioning operates by splitting a large matrix into a number of submatrices of reduced order. These submatrices are consequently more amenable to treatment by both traditional mathematical and numerical tools, as well as advanced parallel processing methods. The use of such methods is usually motivated by the underlying system structure.

High-dimensional system models in engineering and technological fields tend to arise from one of two main sources:

(i) composite systems constructed from a number of (relatively simple) interconnected elements or subsystems
(ii) spatial discretisation of distributed parameter systems whose solution using partial differential equation modelling is either highly complex or intractable.

Often the result in both cases is a very large set of relatively simple relational equations. State-space models have correspondingly sparse state matrices. Consequently, state matrices can often have very regular structures imposed on them by simply grouping equations with the same or similar dependence relations together, or by applying automatic 'clustering' algorithms.[3,4]

Application of such methods can lead to quite compact gathering of non-zero elements around the leading diagonal leading to (nearly) 'block diagonal' or 'banded' forms (see Fig. 2). Matrix partitioning schemes take advantage of such structures, and have proven to be particularly attractive in application to technological systems. The method can be attractive even in sequentially oriented LSS computation. Reasons are plain enough if even the simplest of examples is considered.

If an $(n \times n)$ matrix A can be partitioned (as indicated by the broken line in Fig. 3) into two

$$\left(\frac{n}{2} \times \frac{n}{2}\right)$$

independent submatrices A_1 and A_2, storage requirements are then reduced from $n^2 = 36$ elements to

$$2\left(\frac{n}{2}\right)^2 = 18$$

elements. Furthermore, the possibility of conducting matrix operations on A in terms of independent and concurrent operations on each of the submatrices is introduced.

A number of formal methods for matrix partitioning have been proposed in the literature. Allidina *et al.*,[5] however, propose the use of heuristics on the grounds that completely independent subproblems (such as the submatrices of the previous example) are highly unlikely to arise in practice. Usually, partitioning creates a number of 'coupling' variables between subproblems, which reflect the fact that they originate from a single 'global' problem.

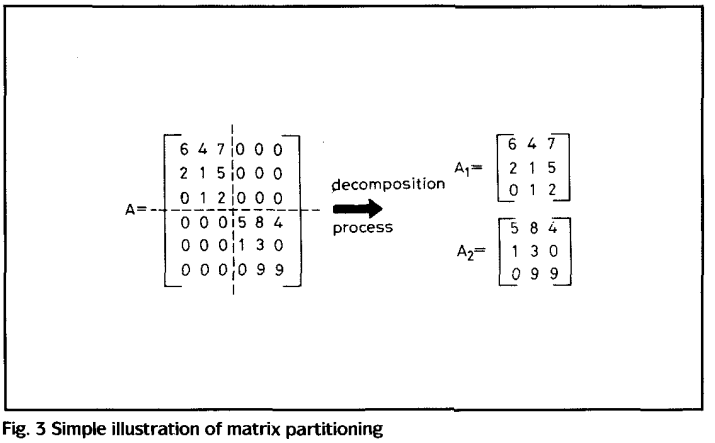

Fig. 3 Simple illustration of matrix partitioning

In cases where coupling variables exist, some form of co-ordination in the evolution of the numerical solution is required, to ensure that results from the new formulation of the problem are consistent with the global one. From considerations such as these 'hierarchical decomposition co-ordination methods' have emerged. Traditionally associated with large-scale optimisation and control problems, these techniques may be conceptualised as a two-level hierarchical structure (see Fig. 4). This structure comprises: a co-ordinator element on the first (upper) level; a number of independent subproblems (which may be solved concurrently) on the second (lower) level; and an iterative information exchange between them. In some instances, further decompositions may be applied, leading to multi-level hierarchies.

In either case some or all of the following objectives should be met by the partitioning process:

(i) The coupling variables should be rather few relative to the overall system dimension in order to ensure numerical stability and to maximise the parallelism in the problem.
(ii) The subsystems should, if possible, be of an equal and moderate computational complexity in order to be able to allocate the workload evenly between processors and to solve each of the subsystems efficiently.
(iii) The couplings should, if possible, be 'weak', although the notion of 'weak couplings' cannot easily and uniquely be expressed quantitatively (see for example Chow and Kokotovic[6] for further discussion).

Clearly, these three objectives can be somewhat contradictory, as is usually the case in multiobjective problems. As a result it would seem that '... it is very difficult, if not impossible, to develop a general purpose automatic (formal) procedure for system partitioning'[5].

Since subproblems and co-ordinator tasks resulting from the matrix partitioning approach are generally heterogeneous (dissimilar) in nature, parallel processing architectures with MIMD capability are most appropriate for their implementation.

Alternatives to aggregation and matrix partitioning methods currently appear rather thin on the ground. One early and tentative proposal was made by Nievergelt[7], which showed that it is possible, at least in principle,

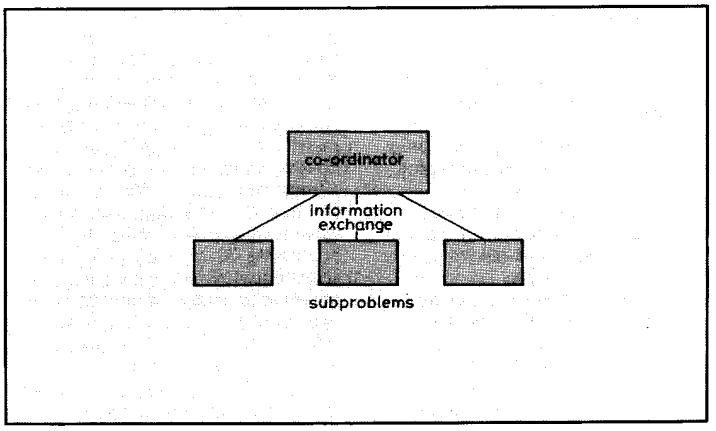

Fig. 4 Schematic illustration of hierarchical decomposition co-ordination methods

to introduce parallelism at the price of computational redundancy by dissection of the interval of integration. Solutions are computed for each subinterval using a range of predicted values at each subinterval boundary. On completion of the integration process, 'best' subinterval solutions are linked by linear interpolation techniques.

A contemporary view of decomposition-in-time is that, 'the method is not particularly effective, even for linear equations'[1]. However, the method can result in computational speed up, although a substantial amount of extra processing may be involved (especially for a nonlinear set of equations).

This author is of the opinion that the fundamental flaw of the approach is due to it not being a 'natural' decomposition. Specifically, the scheme attempts to exploit parallelism within the evolution of the independent variable (time). This is perhaps the only truly sequential aspect of simulation problems.

Discretisation

If attention is now focused on those methods for exploiting concurrency that can be applied during discretisation, then Miranker and Liniger[8] have introduced the notion of a 'parallel front' in computational processes, and have incorporated the property into a number of predictor-corrector and Runge-Kutta numerical integration procedures. The methods are intended for implementation on $2s$ ($s = 1,2,3,\ldots$) processors with MIMD capability. As an aid to understanding the parallel front method, consider the predictor-corrector formulation as an example.

A predictor-corrector pair consists of an explicit predictor formula which provides an initial estimate of the solution subsequently used by the implicit corrector formula. The advantage of the approach is to maintain accuracy and reduce the iterations required compared to the application of a purely implicit formula, while improving over the numerical stability properties of a purely explicit formula.

Conventional formulations of these algorithms have an underlying sequential nature in that the application of the predictor must strictly precede that of the corrector in any one time step. This is because predicted values for the next time step usually require the corrected values at the current time step. In their simplest form, the Miranker and Liniger formulations break this mould by allowing predicted values for the next integration step to be computed at the same time as corrected values at the current integration step. Hence a parallel mode (or front) of integration is introduced.

The use of parallel front methods may be somewhat restricted in practice since studies have shown that the introduction of parallelism results in a marked reduction in the size of the stability regions of such methods[9]. Consequently their use may not be appropriate for stiff systems.

An alternative to parallel front numerical methods are those methods studied by Worland[10] and collectively known as 'block implicit methods'. The word 'block' refers to the fact that a k-point set of new solution estimates are produced at each application of the algorithm. Block-parallel predictor-correctors first predict k discrete values concurrently, then correct k values concurrently.

Use of the methods can be limited by the fact that the required number of function evaluations within each numerical formula increases at a rate proportional to the degree of parallelism introduced[11].

Post-discretisation

Numerical simulation algorithms applied to ODE systems are generally realised as a set of algebraic vector-matrix expressions and the possibility exists to exploit what has been termed 'sub-integration-cycle level' parallelism within an existing discretisation. A number of these post-discretisation methods are available.

One such method is known as 'divide-and-conquer'. In its most general form, divide-and-conquer involves the fragmentation of a problem into smaller subproblems which may be treated independently. A simple example illustrates the principle: If $y = Ax + Bx$ (where A and B are $(n \times n)$ matrices, and y and x are $(n \times 1)$ vectors), the vector-matrix products (Ax and Bx) are tightly coupled and computationally intensive [$O(n^2)$-operations]. Therefore, one approach to the exploitation of coarse-grained parallelism in this problem would be to compute each of these products independently on different processors, then perform the relatively loosely coupled and computationally less demanding [$O(n)$-operation] of adding the two resulting vectors on one of the processors, subsequent to the necessary communication and synchronisation processes.

Notice that the vector y cannot be evaluated until the solutions to both Ax and Bx are available. In other words, the computations must be arranged in a way that does not violate the precedence constraints of the algorithm. Notice also that strict adherence to algorithmic precedence constraints ensures the uniqueness of the numerical results.

A close relative of the divide-and-conquer method is known as 'reordering'. In this technique, parallelism is introduced by appropriate (re)structuring of the algorithmic sequence. Consider the equation $y = (A + B)x$. This equation suggests an algorithm for computing the vector y by first performing the n^2 floating-point additions to form the matrix $(A + B)$ then performing n^2 floating-point multiplications to form y from the vector-matrix product of $(A + B)$ and x. This algorithm implicitly assumes sequential arithmetic processing. An obvious rearrangement of the equation, however, gives the equation used to illustrate the divide-and-conquer concept. A parallel solution strategy for this equation has already been suggested.

Notice though, that the 'sequential' form of the algorithm requires n^2 floating-point multiplications and n^2 floating-point additions. Conversely, the 'parallel' form requires $2n^2$ floating-point multiplications and n floating-point additions, as well as the necessity for synchronisation and data communication. If floating-point multiplication is computationally more expensive than floating-point addition (as is usually the case), then the parallel algorithm is said to have introduced 'computational redundancy'. Hence the meaningful use of reordering strategies implicitly dictates that the gain of concurrency outweighs any overhead of redundancy. This desirable situation is most likely to occur if algorithm and architecture are 'well-matched'.

Another post-discretisation method is that of 'recursive reduction' developed by Kogge[12]. This very general principle is used for exploiting parallelism within algebraic expressions constructed from associative operations. Recursive reduction represents a particular combination of divide-and-conquer and reordering, and is important because of its applicability to a wide range of problems. Once more, the technique is illustrated here by an example.

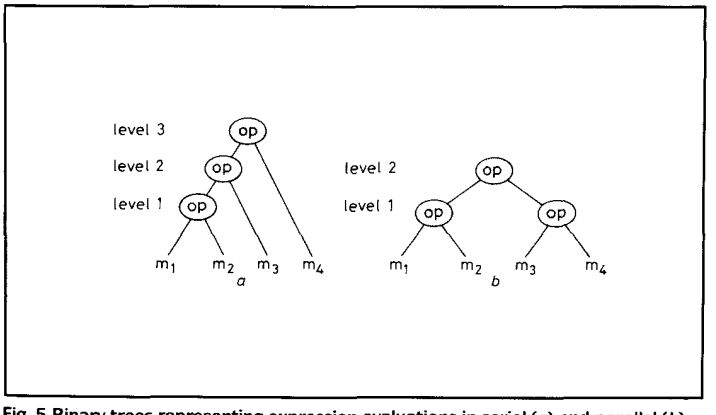

Fig. 5 Binary trees representing expression evaluations in serial (*a*) and parallel (*b*) modes

Consider a set of four elements $\{m_1, m_2, m_3, m_4\}$ linked by the associative operator *op*. The expression $X = m_1\ op\ m_2\ op\ m_3\ op\ m_4$ can be computed in both a serial and a parallel fashion as shown in Fig. 5. Essentially these binary trees represent the precedence constraints for evaluation of the expression x, under differing assumptions about the processing architecture. Notice how the 'height' of the tree is reduced in the case of the parallel (recursive reduction) implementation, signifying a reduction in processing time. Generally, recursive reduction applied to $N = 2^n$ $(n = 1,2,3,\ldots)$ elements, requires $\ln(N)$ parallel steps of computation whilst a sequential implementation requires $N-1$ steps.

At even lower levels of task granularity, such as scalar floating-point or integer arithmetic, the art of parallel numerical analysis has emerged. As a result, parallel versions of many core algorithms (such as matrix inversion) are now available. An excellent introduction to the subject is given in the book by Schendel[13]. The classic review article of Miranker[14] is an extensive survey of the field, whilst those by Heller[15] and Feilmeier[16] monitor the progress of the state of the art.

Integrated approach

Concurrency exploitation before discretisation is mainly linked to the physical and mathematical properties of the system and its model. Concurrency enhancement during discretisation is allied to the incorporation of parallelism into the numerical solution procedure. Concurrency enhancement after discretisation is associated with the exploitation of sub-integration-cycle parallelism of an existing numerical algorithm. Consequently, these approaches are not mutually exclusive! It would appear possible to envisage an integrated strategy, incorporating concurrency exploitation at each of these three levels, applied to a specific applications problem.

The principal tool for pre-discretisation concurrency exploitation appears to be matrix partitioning methods. These methods are characterised by data partitioning and process (algorithm) replication, and may be used:

(i) after the application of clustering algorithms (which help to expose the system structure)
(ii) following aggregation (if the associated loss of model detail is acceptable)
(iii) in conjunction with

decomposition-in-time (if the particular system structure permits).

Exploitation of parallelism within the problem structure may be complemented by additional parallelism derived via judicious selection of a numerical solution procedure.

Following the well known and structured top-down approach, further parallelism may be extracted from the application problem by searching further down the granularity tree and exploiting concurrency within the numerical solution procedure.

As a final remark, it is important to realise that the existence of a parallel algorithm together with the availability of concurrent processing facilities is not sufficient to ensure successful parallel processing implementations. Rather, the principal determinant of performance will be the degree of match between algorithm and architecture[17,18]. These two important elements should not be selected in isolation.

References

1 ZAKHAROV, V.: 'Parallelism and array processing', *IEEE Trans.*, 1984, **C-33**, (1), pp.45-78
2 HOCKNEY, R.W., and JESSHOPE, C.R.: 'Parallel computers' (Adam Hilger Ltd., 1981)
3 LEI, S., ALLIDINA,, A.Y. and MALINOWSKI, K.: 'Clustering techniques for rearranging ODE systems', Control Systems Centre Report No. 635, UMIST, 1985
4 RIAZ, Z.: 'Block diagonal dominance for large-scale systems', M.Sc. dissertation, Control Systems Centre, UMIST, 1987
5 ALLIDINA, A.Y. *et al.*: 'Development of decomposition co-ordination techniques for simulation', Interim Report No.2, Complex Systems Group, Control Systems Centre, UMIST, 1984
6 CHOW, W.W., and KOKOTOVIC, P.V.: 'Time scale modelling of sparse dynamic networks', *IEEE Trans.*, 1985, **AC-30**, (8), p. 714
7 NIEVERGELT, J.: 'Parallel methods for integrating ordinary differential equations', *Comm. ACM.*, 1964, **7**, (12), pp. 731-733
8 MIRANKER, W.L., and LINIGER, W.: 'Parallel methods for the numerical integration of ordinary differential equations', *J. Maths. for Comp.*, 1967, **21**, pp. 303-320
9 FRANKLIN, M.A.: 'Parallel solution of ordinary differential equations', *IEEE Trans.*, 1978, **C-27**, (5), pp. 413-420
10 WORLAND, P.B.: 'Parallel methods for the numerical solution of ordinary differential equations', *IEEE Trans.*, 1976, **C-25**, pp. 1045-1048
11 TAI, H.M., and SEAKS, R.: 'Parallel system simulation', *IEEE Trans.*, 1984, **SMC-14**, (2), pp. 177-183
12 KOGGE, P.M.: 'Parallel solutions of recurrence problems', *IBM J. Res. and Dev.*, 1974, **18**, pp. 138-148
13 SCHENDEL, U.: 'Introduction to numerical methods for parallel computers' (Ellis Horwood Ltd., 1984)
14 MIRANKER, W.L.: 'A survey of parallelism in numerical analysis', *SIAM Rev.*, 1971, **13**, (4), pp. 524-547
15 HELLER, D.: 'A survey of parallel algorithms in numerical linear algebra', *SIAM Rev.*, 1978, **20**, (4), pp. 740-777
16 FEILMEIER, M.: 'Parallel numerical algorithms', *in* 'Parallel processing systems', (Cambridge University Press, 1982)
17 YALAMANCHILI, S., and AGGARWAL, J.K.: 'Reconfiguration strategies for parallel architectures', *IEEE Computer*, 1985, **18**, (12), pp. 44-61
18 BURTON, A.W.: 'A methodology for the simulation of large-scale dynamical systems using concurrent computing facilities', Ph.D. Thesis, Control Systems Centre, UMIST, 1989

Dr. Anthony Burton is Projects Research Officer, Advanced Controls Department, Lucas Automotive Ltd., Advanced Engineering Centre, Dog Kennel Lane, Shirley, Solihull, West Midlands B90 4JJ, UK. He is an IEE Associate Member.

CALENDAR

This is a free listing. Please send details of events for possible inclusion to the Managing Editor, *Computing & Control Engineering Journal*, IEE, Michael Faraday House, Six Hills Way, Stevenage, Herts. SG1 2AY, UK.

15th March 1991
Colloquium on CSCW: Some Fundamental Issues, London. Contact: Computing & Control Division, IEE, Savoy Place, London WC2R 0BL, UK.
Tel: 071-240 1871

17th-20th March 1991
Third Conference on Telecommunication, Edinburgh, UK. Contact: IEE Conference Services, Savoy Place, London WC2R 0BL, UK.
Tel: 071-240 1871

18th March 1991
Colloquium on Systems and Applications of Man-Machine Interaction Using Speech I/O, London. Contact: Computing & Control Division, IEE, Savoy Place, London WC2R 0BL, UK.
Tel: 071-240 1871

19th March 1991
Colloquium on Discrete Event Simulation—a Decision Making Process, not just Computer Modelling, London. Contact: Computing & Control Division, IEE, Savoy Place, London WC2R 0BL, UK.
Tel: 071-240 1371

19th-21st March 1991
AIS 91—Advanced Information Systems, London. Contact: Jean E. Mulligan, Learned Information, Woodside, Hinksey Hill, Oxford OX1 5AU, UK.
Tel: 0865 730275, Fax: 0865 736354

21st-22nd March 1991
Two-day Symposium on Robust Control System Design Using H-Infinity and Related Methods, University of Cambridge. Contact: Institute of Measurement & Control, 87 Gower Street, London WC1E 6AA, UK.
Tel: 071-387 4949

25th-28th March 1991
International Conference on Control '91, Edinburgh. Contact: Conference Services, IEE, Savoy Place, London WC2R 0BL, UK. Tel: 071-240 1871

26th-28th March 1991
International Conference on Concurrent Engineering and Electronic Design Automation, Bournemouth, UK. Contact: Conference Services, IEE, Savoy Place, London WC2R 0BL, UK.
Tel: 071-240 1871

4th April 1991
Colloquium on Integrating Issues in Aerospace Control, London. Contact: Computing & Control Division, IEE, Savoy Place, London WC2R 0BL, UK.
Tel: 071-240 1871

5th April 1991
Colloquium on Advances in Supervisory Control and Data Acquisition (SCADA) Systems, London. Contact: Computing & Control Division, IEE, Savoy Place, London WC2R 0BL, UK.
Tel: 071-240 1871

7th-11th April 1991
INFOCOM '91, Bal Habour, FL, USA. Contact: Ken Joseph, Bell Canada, 160 Elgin Street, Ottawa, Ontario, K1G 3J4, Canada.
Tel: (613) 781-7214

8th-9th April 1991
1st International Workshop on Interoperability in Multidatabase Systems, Kyoto, Japan. Contact: IEEE Computer Society, Conference Services, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903, USA.
Tel: (202) 371-1013

8th-10th April 1991
Vacation School on Network Technology, Swansea, UK. Contact: IEE Conference Services, Savoy Place, London WC2R 0BL, UK.
Tel: 071-240 1871

8th-12th April 1991
Vacation School on Systems Engineering, Reading, UK. Contact: IEE Conference Services, Savoy Place, London WC2R 0BL, UK.
Tel: 071-240 1871