

PARALLEL PROCESSING TECHNIQUES FOR FE ANALYSIS: STIFFNESSES, LOADS AND STRESSES EVALUATION

Dimitris Zois⁺

Department of Aeronautics, Imperial College, Prince Consort Road, London SW7 2BY, U.K.

(Received 18 December 1986)

Abstract--As new computer architectures emerge using parallel processing methods for more processing power, the need to explore and exploit parallelism in Finite Element Analysis, a numerical method that uses large amounts of calculations, arises naturally. In this work we present parallel processing techniques for the generation of element stiffnesses, element loads, the assembly process and the evaluation of the element stresses. These techniques can be used for the application of the Finite Element Method for structural analysis work on parallel computers. PARFES, a Parallel Finite Element System, which implements these techniques on particular parallel computers, is discussed. Our target parallel computer is a Multiple Instruction Multiple Data type that follows the Communicating Sequential Processes model of computation. The parallel programming language OCCAM is used for the current implementation of PARFES and our main objectives, apart from using parallel processing, are portability and adaptability of the system among MIMD parallel computers that follow the CSP model of computation.

INTRODUCTION

Although the FEM can be traced back many years [1] as a mathematical theory, its growth and real potential as a practical engineering tool was realised with the development of modern digital computers. It is natural therefore that the large scale FEM computer programs (FEM Systems) proliferated during the early 1970s [1,2], using the most advanced properties of digital computers of that time. This implies that these FEM systems were, and still are, using a serial model of computation as are the computers on which the systems have been implemented. This computation model is attributed to one of the pioneers of computing, namely John von Neumann [3], and can be briefly described as the computer which has one processor and memory associated with it. This computer configuration, for each single time instant, delivers one operation on a pair of data.

In the late 1960s and early 1970s, the general purpose serial computers such as CDC 6600 or IBM 360/91 [4] began to use multiple dedicated functional units, e.g. floating point adders, multipliers, working in parallel to improve the speed of the program execution. This form of parallelism was actually hidden from the user who was working with a high-level programming language (e.g. FORTRAN) to write his programs. The only way to directly exploit this parallelism was by assembly language programming. However, the FORTRAN compiler on these computers would optimize a program to utilise these facilities.

From the very beginning of the construction and use of digital computers, several researchers have proposed various forms for parallel configuration employing different ways of parallelism [5]. Without going into the hardware or software details, a classification scheme, now

widely accepted, was proposed by Flynn [6, 7]. According to this classification the computer domain may be divided into four broad categories which are distinguished from each other by the number of instruction Streams and Data Streams that a computer handles at a particular time instant. These categories are:

SISD: Single Instruction Single Data, which represents the usual serial computers.

MISD: Multiple Instruction Single Data, which represents some of the so called pipelined computers.

SIMD: Single Instruction Multiple Data, which represents processor arrays.

MIMD: Multiple Instruction Multiple Data, which represents multiple processor computers.

More details about the above mentioned categories and the computers belonging to each of them, as well as other classification schemes, may be found in [5].

The basic idea behind the introduction of parallel computers (or supercomputers as some of those are called [8]) is: if one processor needs time T to solve a given problem, then P processors should need only time T/P for the same problem. If the cost to introduce a parallel computer with P processors is less than P times the cost of a single processor computer then significant speed increase and cost reduction can be obtained for the computer solution of a given problem. Furthermore, because of the physical limits that a single processor computer can not exceed (e.g. speed of light for signal propagation) and the increasing demand for more calculations [9], the need to use more than one processor cooperating on the solution of a problem emerges naturally [5].

⁺ Present address: University of Patras, P.O. Box 1164, GR-261 10 Patras, Greece.

Almost concurrently with the appearance of the first computers which use parallelism (e.g. ILLIAC IV, CDC STAR 100, Cray 1, etc.), many proposals and research about the use of these machines for the programming of the FEM have been published [10-20]. For a more complete survey of these efforts see [21].

The main problems that should be considered for the successful exploitation of current and future parallel computers for FE analysis work are:

(a) Software portability, i.e. the FEM system should not be redesigned for a new parallel computer.

(b) Software adaptability, i.e. no need to rewrite a FEM system to accommodate the increase of the number of processors in a particular parallel computer.

To successfully address the above problems, we must concentrate our attention on one category of parallel computers and on one computational model for that category. The different possibilities of parallel computation (computational models) will be discussed later in this paper. Here and in [22], we propose a Parallel Finite Element System (PARFES) for MIMD type parallel computers, which support the Communicating Sequential Processes (CSP) model of parallel computation [23, 24].

The selection of the MIMD type parallel computers was made because we think that it represents the most general case of parallelism and it can emulate most of the other existing types. The CSP computational model was selected because we think that it is a simple, elegant and powerful model; and when used for parallel processing work, it yields clear and concise results which, we believe, are required to address effectively current and future parallel computation needs. Furthermore, this computation model of MIMD type parallel computing is considered to offer unrestricted expansion in the number of processors that a parallel computer may employ, providing that an expandable interconnection scheme between processors has been selected.

THE FE DISPLACEMENT METHOD

The Finite Element Method is a general numerical technique for the solution of partial differential equation systems, subject to appropriate boundary and initial conditions. We shall present here briefly the method as used for the solution of structural analysis problems [25]. A more complete presentation may be found elsewhere (e.g. [26]).

In the FE displacement method it is assumed that the displacements within an element may be described in terms of the displacements at the element's nodes as:

$$[\delta] = [N] * [\delta^e] \tag{1}$$

where

$[\delta]$ = vector of displacements within an element

$[N]$ = matrix of the interpolation or shape functions

$[\delta^e]$ = nodal displacements of element.

The strains within the element may then be expressed as:

$$[\epsilon] = [B] * [\delta^e]$$

where

$[\epsilon]$ = vector of strains

$[B]$ = matrix consisting in general of the shape functions and their derivatives.

Finally the stresses may be represented as:

$$\sigma = D \epsilon$$

or

$$\sigma = D \delta$$

using (2) and (3), where

$[\sigma]$ = vector of stresses

$[D]$ = stress-strain relationship (constitutive law).

If we form the expression that gives the potential energy of the element and take the first derivative of the energy with respect to the nodal displacements we have [25]:

$$\int ([B] * [D] * [B]) * [\delta] * dV - \int [N] * p * dV - \int_{S_e} [N]^T * q * dS = [K] * [\delta] - [F] = [] \tag{5}$$

where

p = body forces for the element

q = surface forces for the element

V_e = volume of the element

S_e = surface area of the element

$[K^e]$ = stiffness matrix of the element

$[F^e]$ = equivalent nodal loads for the element.

The potential energy derivative is equal to zero in (5) since the potential energy in the element should be a minimum. The total potential energy for the structure can be obtained by adding the individual energies of each element for all the elements in the structure. By employing the same technique as above we may derive a similar expression as (5) for the whole structure:

$$K \delta = F \tag{6}$$

where

[K] = stiffness matrix of structure

[] = vector of nodal displacements of the structure

[F] = vector of equivalent nodal loads of the structure.

The standard computation procedure for FEM static analysis of a structure is as follows:

(a) Evaluation of the stiffness for each element of the structure and assembly of that stiffness to the appropriate positions in matrix [K] of (6).

(b) Evaluation of equivalent nodal loads for each element of the structure and assembly of these loads to the appropriate positions in vector [F] of (6).

(c) Solution of the system of linear equations of (6) to obtain the unknown nodal displacements.

(d) Evaluation of the stresses within each element of the structure by substituting the known nodal displacements into (4).

In this paper we shall examine (a), (b) and (d) in more detail, while (c) is discussed in [22].

From (5) we have

$$[K^e] = \int_{\Omega_e} ([B] * [D] * [B])^* dV \tag{7}$$

$$[F^e] = \int_{\Omega_e} [N]^T * p * dV + \int_S [N] * q * dS \tag{8}$$

Furthermore we may write

$$[\delta^e]^T = [\delta_1 \ \delta_2 \ \dots \ \delta_i \ \dots \ \delta_k] \tag{9}$$

$$[N] = [N_1 \ N_2 \ \dots \ N_i \ \dots \ N_k] \tag{10}$$

$$[B] = [B_1 \ B_2 \ \dots \ B_i \ \dots \ B_k] \tag{11}$$

$$[K^e] = \begin{bmatrix} K_{11} & K_{12} & \dots & \dots & \dots & K_{1k} \\ K_{21} & K_{22} & \dots & \dots & \dots & K_{2k} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & K_{ij} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ K_{k1} & K_{k2} & \dots & \dots & \dots & K_{kk} \end{bmatrix} \tag{12}$$

$$[F^e]^T = [F_1 \ F_2 \ \dots \ \dots \ F_i \ \dots \ \dots \ F_k] \tag{13}$$

$$[\sigma^e] = [\sigma] + [\sigma] + \dots + [\sigma_i] + \dots + [\sigma_k] \tag{14}$$

where

δ_i = the displacement at node i

N_i = the i th shape function of the element defined by δ_i

B_i = the i th matrix that relates the strain within the element to the displacement of the corresponding node of the element

K_{ij} = the stiffness that couples nodes i and j

F_i = the equivalent nodal loads on node i

σ_i = the stresses in an element due to displacements δ_i .

If we substitute (11) and (12) in (7) and do the appropriate matrix operations, we shall obtain

$$K = \int [B] * [D] * [B] * dV \tag{15}$$

In the case that each node of the element has more than one degree of freedom, then (15) will give a matrix of the same order as the number of freedoms.

If we substitute (10) and (13) in (8) we have:

$$F_i^e = \int_{V_e} [N]_i^T * p * dV - \int_{S_e} [N]_i^T * q * dS \tag{16}$$

Again, if each node of the element has more than one degree of freedom then (16) will give a vector of the same order as the number of freedoms.

If we substitute (9), (11) and (14) in (8) we have

$$[\sigma] = [D] * [B] * [\delta_i] \tag{17}$$

Using (14), after the evaluation of the participation (8) of all the element's nodal displacements, we can evaluate the stresses in the element.

PARALLEL COMPUTATION MODELS

Various parallel computation ideas have been used or proposed for FE analysis [11-21]. The general conclusion drawn from these proposals is that a new FE program must be designed for each different model [17,20]. The reason is that different types of parallel architecture have been implemented on each parallel computer, so specific program design is needed for the efficient use of the hardware [21].

If we want to effect a particular MIMD parallel computation model and the appropriate parallel algorithm efficiently, then minimization of computation time, storage, communication and waiting time (synchronization) should all be achieved for the (13) solution of a problem. This implies that symmetric situations should be considered for computation and communication, i.e. all the existing hardware must be kept occupied as much as possible, during the execution of the parallel program, doing useful work. Whilst we can predict in most cases the computational complexity of an MIMD parallel algorithm (number of operations), the communication complexity is more difficult to identify. Furthermore, it is the current belief that communication is more expensive than computation [27]. Therefore, when

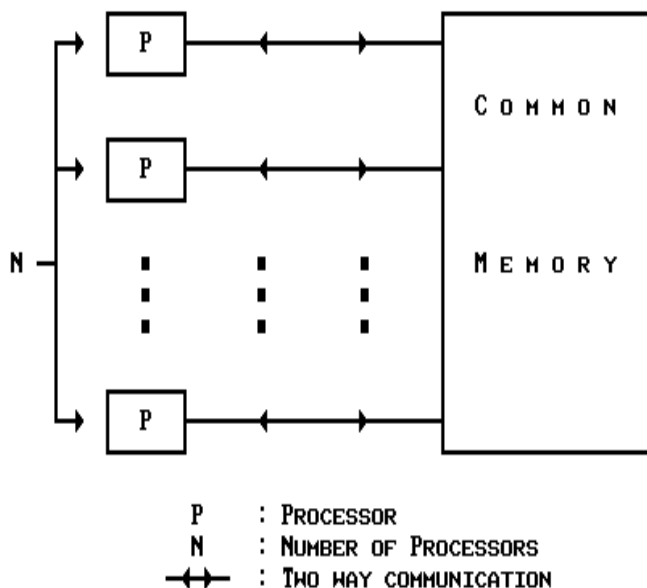


Fig. 1. Shared-variables model.

a parallel algorithm is designed, minimization of communication is perhaps more desirable than of computation or storage.

There are, in general, two different parallel computation models that may be used to describe an MIMD system. Their difference lies in the way that the communication is done between processors.

(1) Communication using shared variables between processors. Usually a common memory is utilized, keeping variables that all processors can read or modify (Fig. 1).

(2) Communication using message-passing between processors. In this case the communication is utilized with channels which connect processors and pass data from one processor to another (Fig. 2).

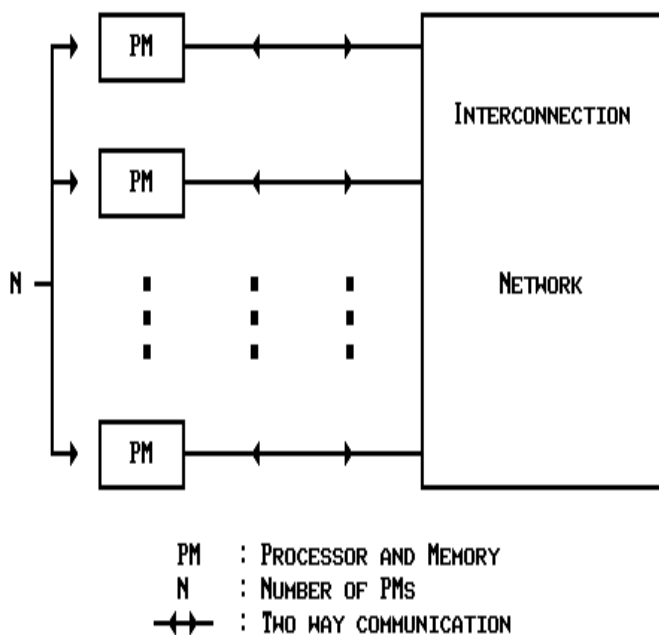


Fig. 2. Message-passing model.

In the former method of communication conflicts may arise if two or more processors try to gain access to the same variable simultaneously. Synchronization mechanisms are therefore employed to prevent such situations. Of course, in that way, the total execution time for a program will increase. The time increase, due to synchronization, is considered prohibitive to the design of parallel computers sharing the same memory if more than a relatively small number of processors are used (i.e. 8-10 processors). This restriction does not apply to message-passing parallel computers because there is not a shared facility between all processors. Therefore message-passing parallel computers are constructed more easily and offer greater expansion potential. For these reasons improved performance can be obtained with reduced cost [28, 29].

The communication between two processors in a message-passing environment may be implemented as:

- (1) Synchronous, where communication proceeds when both processors are ready to exchange data.
- (2) Asynchronous, where a processor is permitted to send data whenever it is ready to do so, regardless of whether or not the other is ready to receive.

The main disadvantage of synchronous communication is that one of the processors may wait idle for the other to be ready, while in asynchronous communication sufficient temporary storage for the exchanged data must be provided, which is expensive and in most cases not practical because the size of the exchanged data, in most cases, is not known beforehand.

In a message-passing computer, the processors are usually linked in a particular interconnection scheme. The most general scheme is a full connection where each processor is connected to all others, while the most restricted is a ring connection (Fig. 3). In practice, a full connection scheme is unlikely for a large number of processors, because of the physical and economical constraints [28]. In a real MIMD parallel computer a partial connection scheme will probably be used. In the case that two processors which are not directly connected have to exchange data, the communication must be done by establishing an indirect connection path. Therefore, each processor must have the capability (hardware or software) to redirect data, to be delivered to all others that are directly connected to it. This indirect communication capability is essential, when a partial connection scheme is employed, in order to substitute the communication facilities that a full connection scheme can provide. A distinction between processor and process must be made here. Whilst a processor represents the actual hardware that is doing operations according to given instructions (i.e. microprocessor), process is a software concept representing a self-contained program that starts, performs some actions, and stops or terminates.

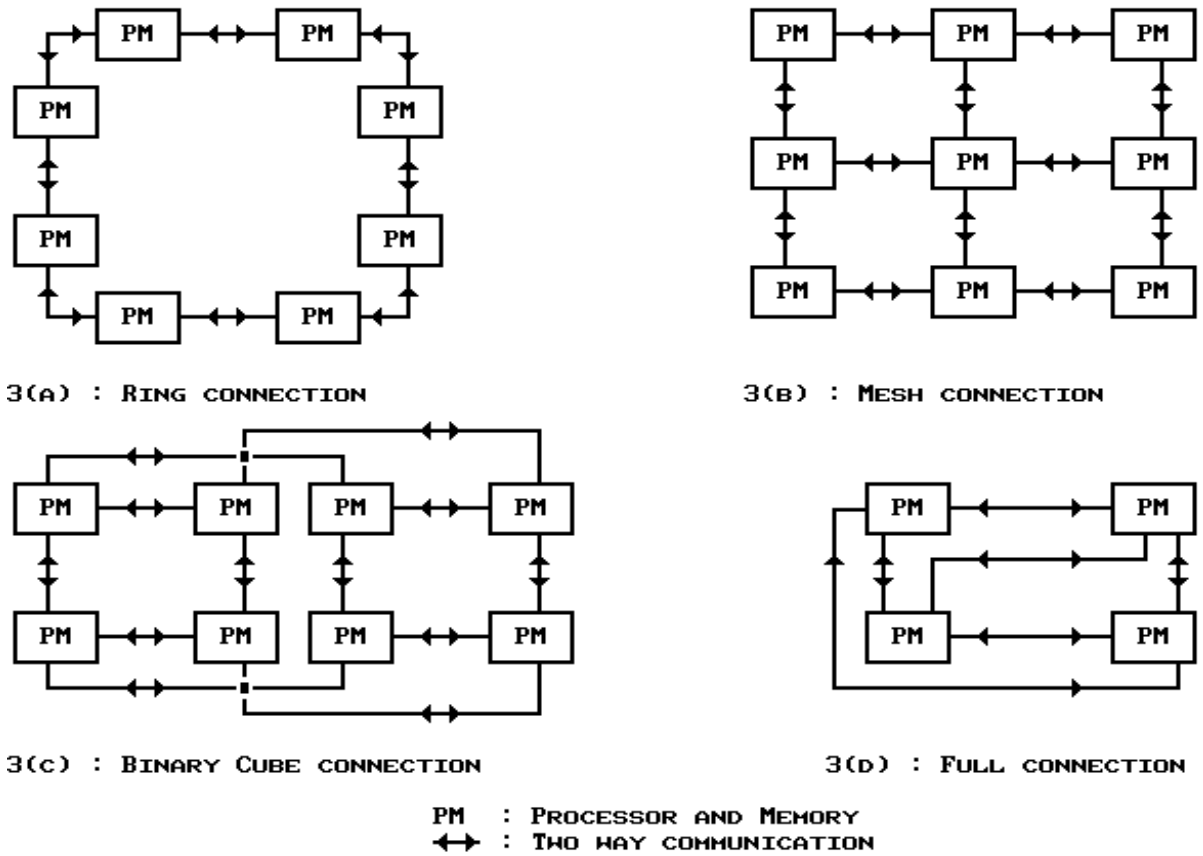


Fig. 3. Examples of processor interconnection schemes.

In practice we may have one or more processes executed by each processor. In the latter case, and if the execution of these processes must be done in parallel, a mechanism must exist so that the processor can execute all processes in turn, giving each a fair share of the time, until all processes have finished. Execution of a program like that (more than one processes executed in parallel by one

processor) is usually called concurrency [30] (Fig. 4). In this work a message-passing, synchronous communication parallel computation model is adopted. This model is of CSP type as implemented in OCCAM [24, 31]. OCCAM is a new programming language that can be used for programming parallel systems.

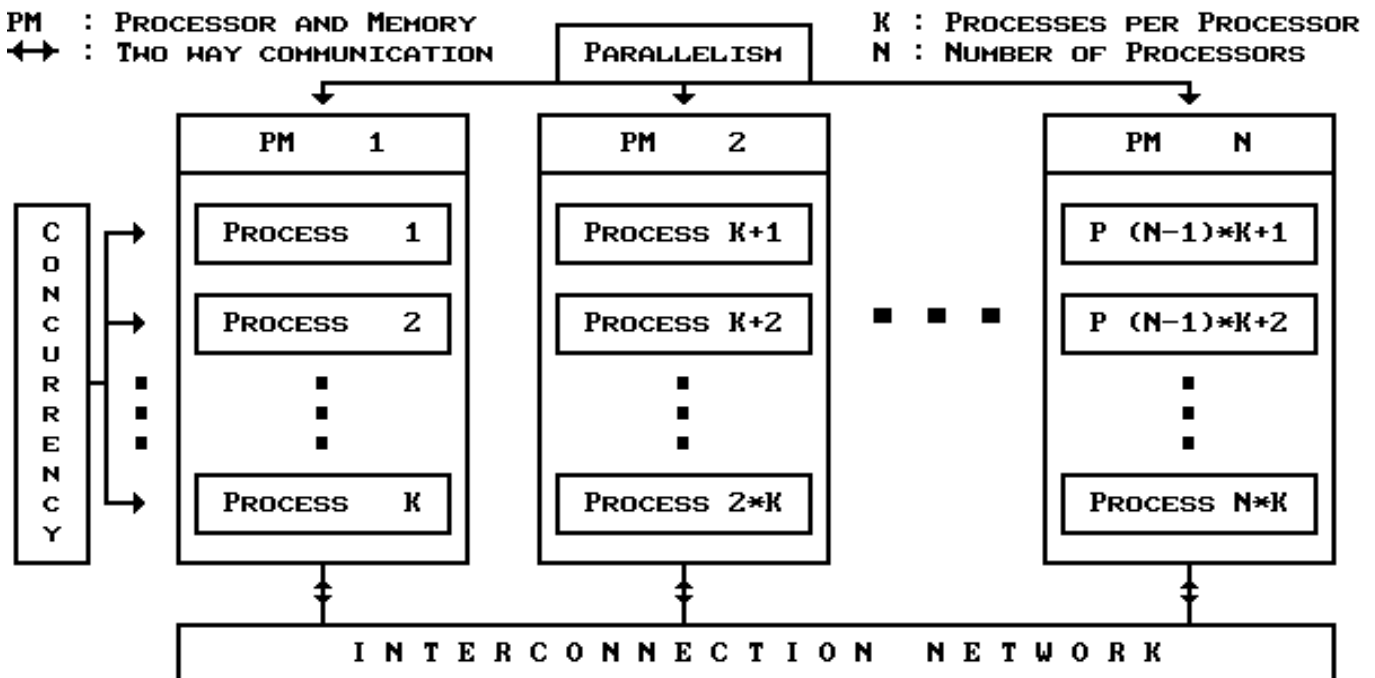


Fig. 4. Parallelism and concurrency.

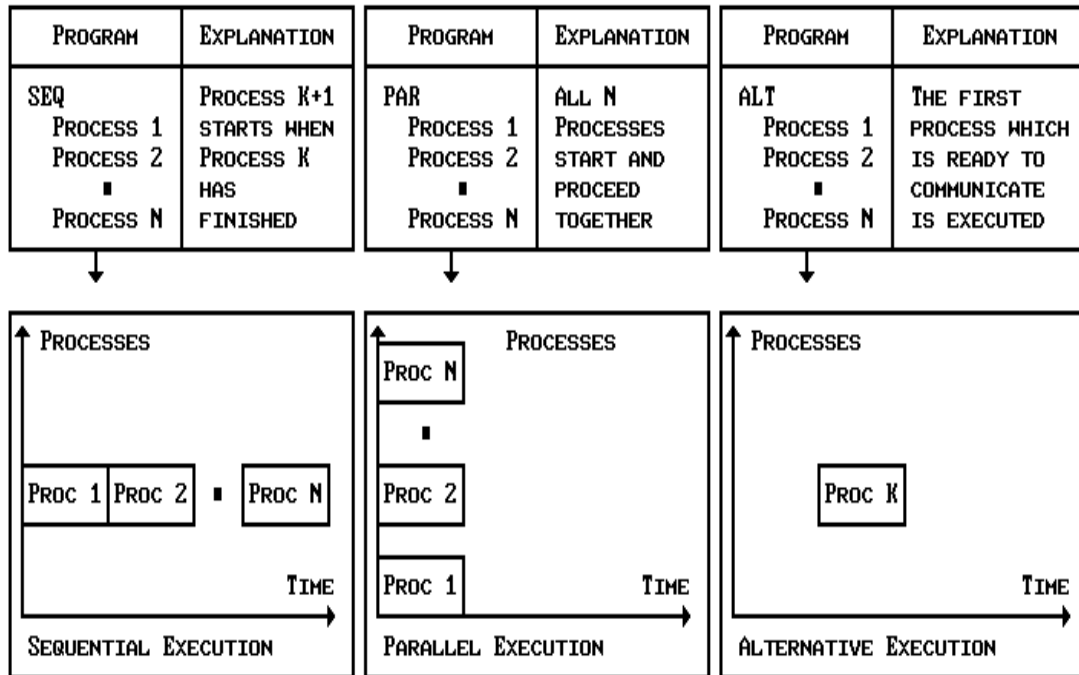


Fig. 5. SEQ, PAR and ALT OCCAM constructs.

A complete description of OCCAM can be found in [31]. Here only a brief presentation is given.

OCCAM

A program in OCCAM can be described as a collection of processes. These processes may be organized for execution in three main ways:

1. Sequentially, i.e. one after the other in time.
2. Parallel, i.e. all together in time.
3. Alternatively, i.e. the first which is ready to communicate will execute.

Any combination of the above three ways may be used in one program (Figs 5 and 6). In OCCAM programming notation each of these main organizations or constructs as

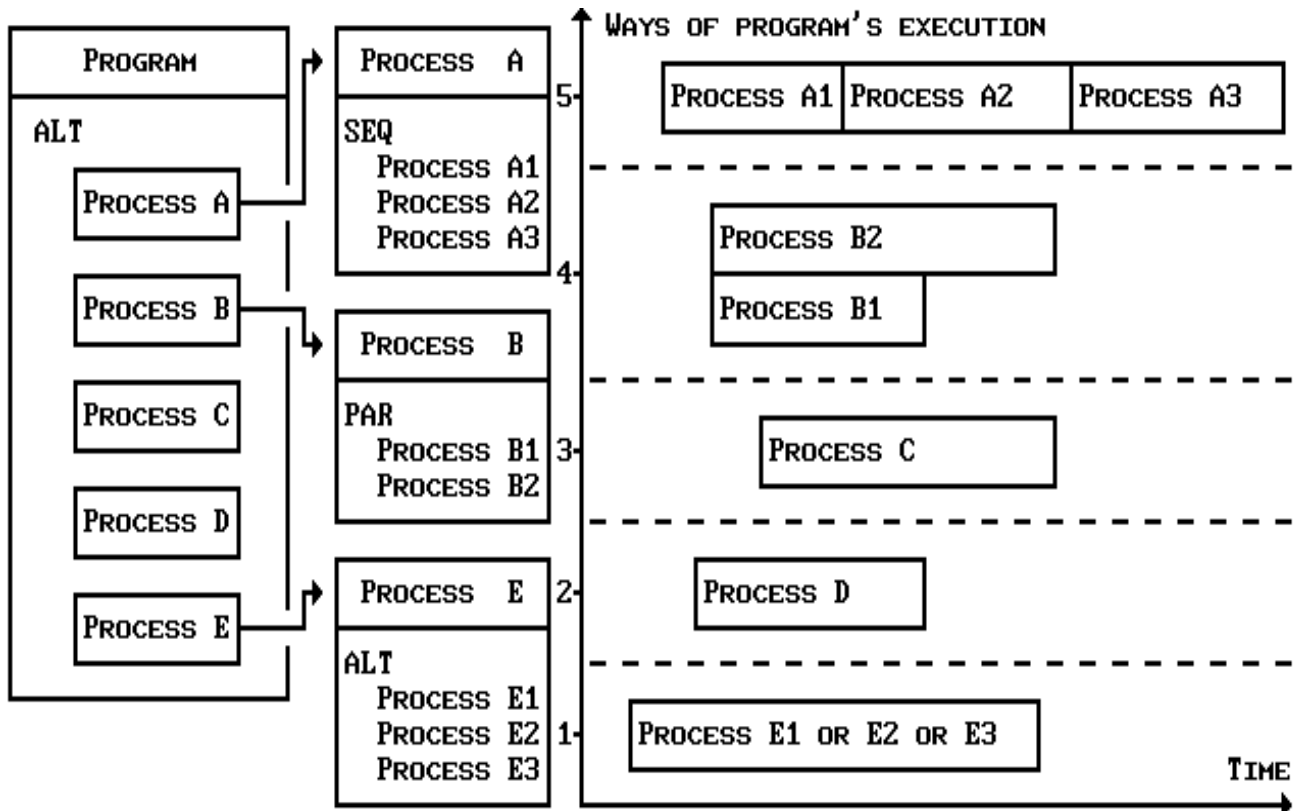


Fig. 6. Combination of SEQ, ALT, PAR in one program.

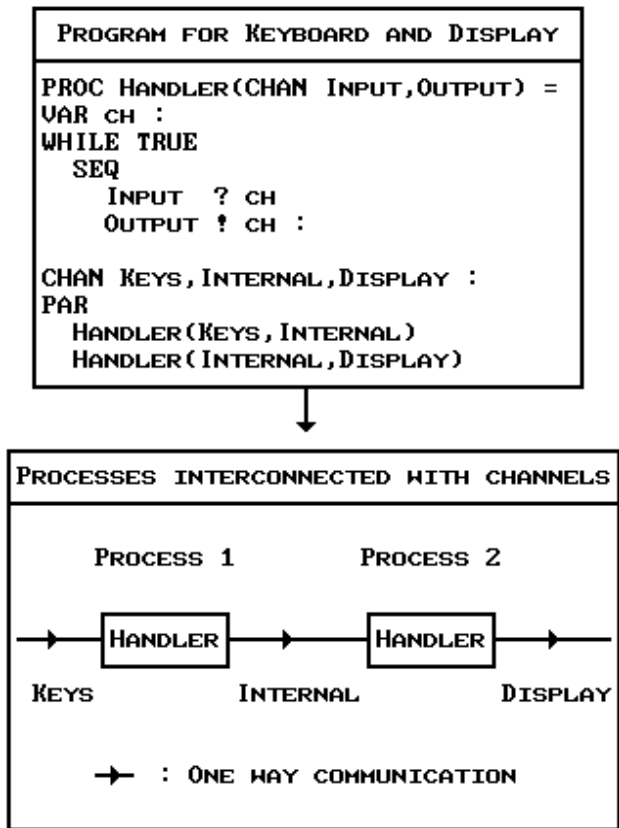


Fig. 7. Parallel program in OCCAM

they are called is represented by a keyword as follows:

- SEQ: sequential execution
- PAR: parallel execution
- ALT: alternative execution.

Furthermore, any process may include other processes, structured appropriately in any depth by SEQ, PAR or ALT constructs. A completely sequential or completely

parallel program can be constructed, or anything between these two extremes.

Other processes, such as more primitive ones, e.g. input, output, assignment etc., or more conventional ones, e.g. repetition (WHILE) or conditional (IF), are included in OCCAM. Communication between processes may be achieved by the use of one-way named communication channels that can be declared before any process, in the same way as variables are declared in conventional programming languages. If we have two processes running in parallel, exchange of data between them is obtained by using only the appropriate channels (Fig. 7).

A parallel program written in OCCAM can run, without changes in its source form, on one or more processors.

When a single processor computer is used for the execution, the parallel processes run concurrently, sharing the time of the processor. The communication channels that connect the processes are, in this case, special positions in the memory (RAM) of the single processor computer. The same execution mechanism that permits the concurrent execution of the parallel processes ensures that the use of channels is in accordance with the rules of the language.

When the parallel program is executed on a multiple processor computer an additional part describing the configuration must be written. The configuration part must be defined before the parallel program and describes the actual parallel hardware on which the program will run, i.e. number of processors, interconnection scheme, and the loading pattern of the parallel program to the multiple processor computer. Furthermore, it allocates processes to processors and assigns the communication channels defined in software to the corresponding hardware (Fig. 8).

A parallel program running on a multiple processor computer will have, in general, some processes allocated

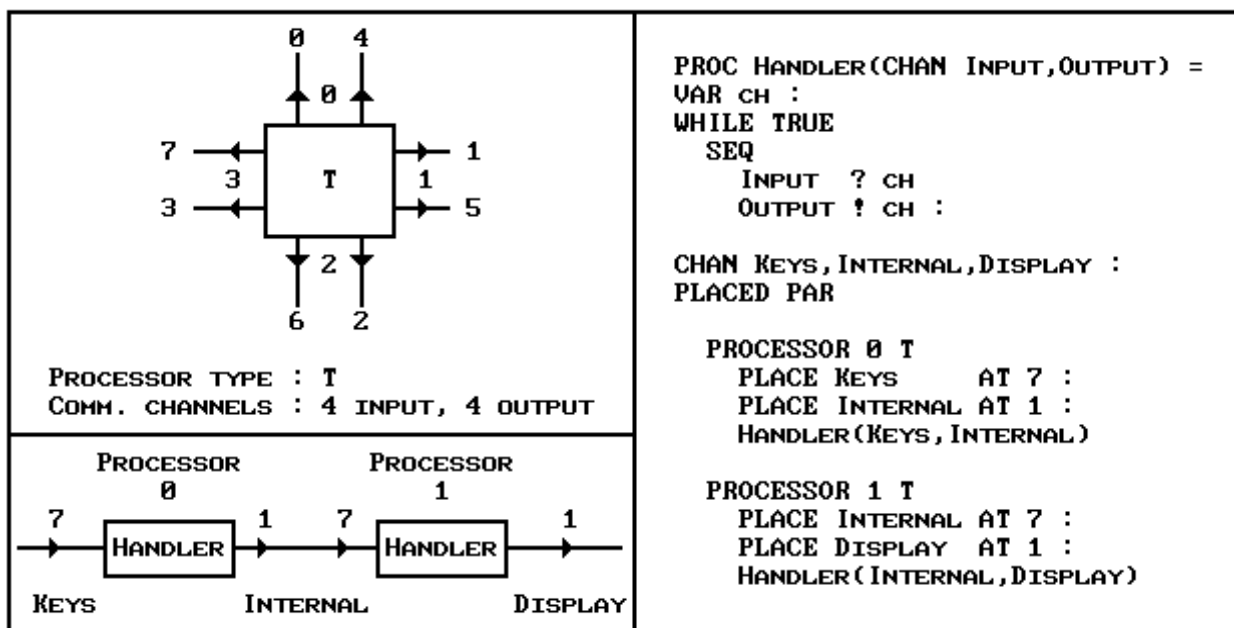


Fig. 8. Configuration example.

to each processor of the computer. The processes that reside in one processor execute in parallel with the processes in any other processor but sequentially or concurrently inside this one processor (Fig. 4). The communication channels that connect processes executing on different processors are mapped with the aid of the defined configuration to actual hardware processor connection lines while the channels that connect processes executing on the same processor are mapped in memory positions as described above for the single processor computer.

The normal way to develop a parallel program in OCCAM is to use the language to write and then execute it on a single processor computer. This approach should be used in order to test the logic of the parallel algorithm. When the results are satisfactory the same program can be executed on a particular multiple processor computer so the performance of the algorithm can be assessed.

If a general purpose algorithm can be devised which can be adopted to all the interconnection schemes from the most restricted (ring) to the most complete (full), then the software will not need to be rewritten to run on a parallel computer with a particular interconnection between processors. It will only be necessary for a new configuration to be written that describes the particular parallel computer on which the program is to be executed. Furthermore, if the number of processors for a particular parallel computer is increased, the program will run unchanged on the expanded computer, provided that the program can accommodate the expansion and the appropriate configuration statements are modified. The above remarks imply that for a given expandable interconnection scheme, larger problems always can be addressed.

Given that the same program can run unchanged on one or more processors, a more general speedup can be defined as:

$$\text{general speedup} = \frac{T}{T_p} = S_p \quad (18)$$

where

T = time for parallel program to run on a computer
with 1 processor

T_p = time for that program to run on a computer
with P processors.

This general speedup, we believe, is more meaningful for MIMD parallel computers than for serial computers; for the usual definition of speedup see [32]. Furthermore, efficiency may be defined as in [32]:

$$\text{efficiency} = \frac{S_p}{P} = \frac{T}{P T_p} = E_p \quad (19)$$

PARALLEL GENERATIONS AND ASSEMBLY

For the solution of the problems in FE analysis (static analysis, displacement method) on MIMD parallel

computers an appropriate partitioning scheme should be selected. So far we have selected a parallel computation model, which complies with CSP [24] as implemented in OCCAM [31]. Therefore we have to restrict ourselves to the adoption of an algorithm that will conform to the above model and can be used on any possible interconnection scheme, from the most simple (ring) to the most complete (full). This restriction is useful because it will permit us to use the same software on MIMD parallel computers with different interconnection schemes. However, we expect differences in speedup, efficiency and performance in general for each of the above computers. By studying different interconnection schemes, the best performance may be obtained for a particular solution method of the FE analysis. Finally, if software reconfigurable interconnection schemes are provided [33] in a parallel computer, perhaps a particular connection which will give the best performance can be found for a particular structural FE analysis without drastic changes in the source code of the FE program.

Almost everyone that has worked on parallel techniques for the solution of the FE problems has observed that generation of the element stiffness matrices and the element nodal loads, as well as the element stresses' recovery, always can be performed in parallel. If such a parallel generation and recovery scheme is used, the highest possible efficiency may be obtained. However, problems arise in the assembly process. Namely, after the element stiffness matrix and nodal loads vector have been generated, they must be added to the appropriate places in the stiffness matrix and load vector for the structure. In the general case, generation and assembly are performed on different processors, so that quite a large amount of communication will be involved to transfer the generated data to the appropriate assembly positions. These positions depend on the numbers assigned on the nodes of the FE mesh and the connectivity of the elements. Occasionally it might be possible to adopt a numbering scheme for a particular FE mesh such that generation and assembly can be done by the same processor. But the employment of such a scheme will not be practical in general. This approach of generation and assembly should not be followed if maximum efficiency is desired. Furthermore, an element by element generation and assembly may leave processors idle because of dissimilar complexities between elements if different types of elements are used in the same FE mesh. The same is also true for the different types of applied loads.

Another way of generation and assembly is therefore proposed here. The task of generation and assembly of each row of the stiffness matrix and of the load vectors from the contributing elements of the structure using (15) and (16) is assigned to one of the processors of the parallel computer (Fig. 9). In this way the complete stiffness matrix is constructed without any need of communication between processors. In the proposed generation and

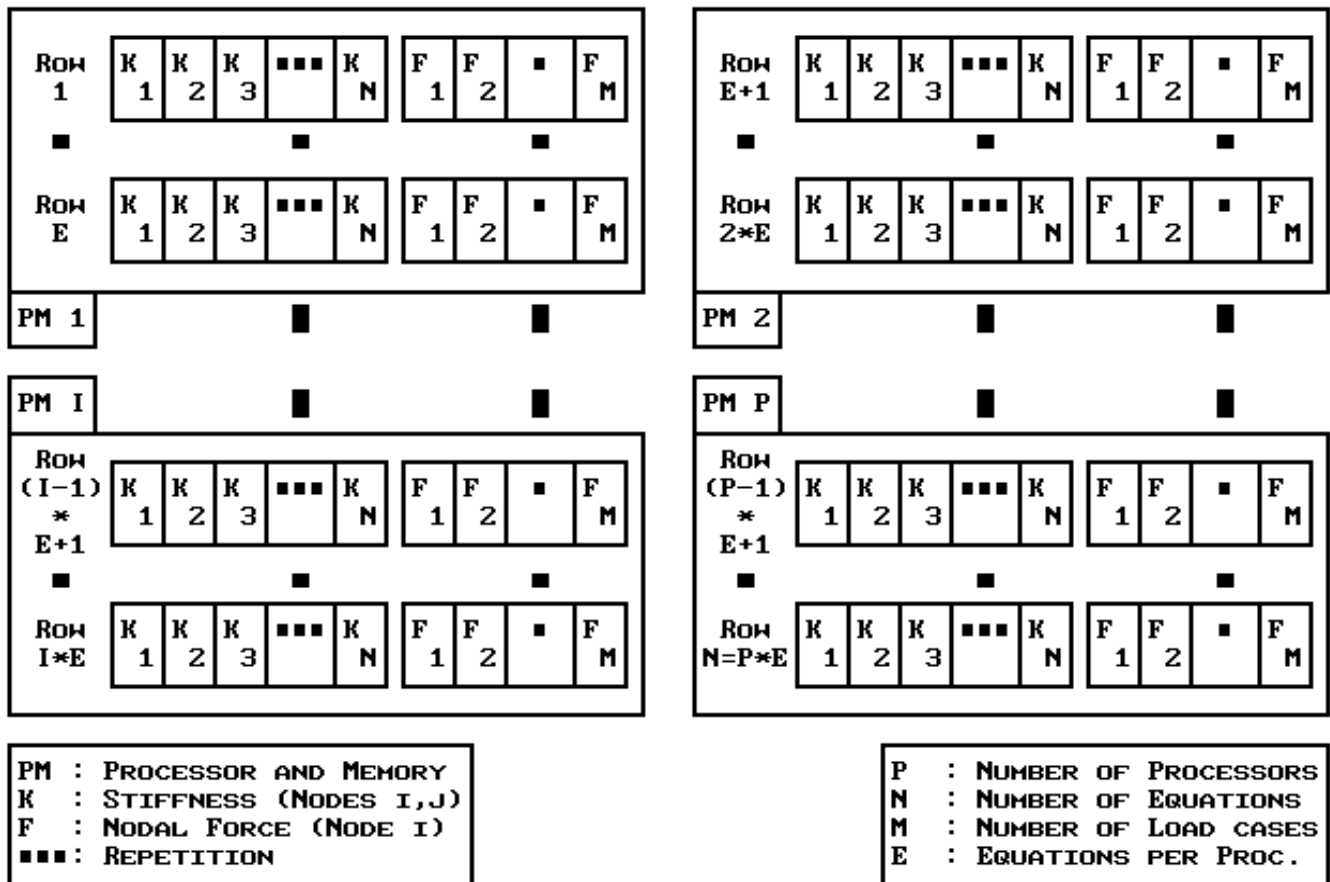


Fig. 9. Generation by rows.

assembly, care must be taken if either of the following two situations arise:

- (1) If the assembly positions of a complete element reside in one processor the generation and assembly should be done element by element using (7) and (8), and not row by row of the stiffness matrix.
- (2) In general, if both K_{ij} and K_{ji} are in the same processor, only one of them should be actually calculated and copied to the position of the other.

Due to the fact that symmetry is only partially used during generation and assembly but not in the storage of the generated matrix, an increase in computation time and storage results for each processor but no co-ordination time (synchronization and communication) is spent at all. Furthermore, the number of rows assigned to each processor can be calculated so that similar computation complexities may result, if necessary. This means that for the generation and assembly processes all processors will spend almost the same time, so there will not be any processors idle for long, waiting for the other to finish. This asynchronous stiffness and loads generation, we think, is more efficient than the one where a processor generates an element stiffness and loads and then sends the generated data to another to be assembled.

The number of rows to be assigned to a processor so that similar computation complexities will result during generation and assembly can be found from:

- (1) The nodal valency of the FE mesh, i.e. number of elements connected to each node.
- (2) The type of the elements that are used.
- (3) The type of loads that are applied.

The above three factors define precisely the computational complexity for each particular row of the stiffness matrix and of the load vectors for a structure during the generation and assembly stages. Consequently we can assign similar computational complexity to each processor, for these stages, even in the case that different elements are used in the same FE mesh.

PARALLEL STRESSES EVALUATION

Since each row of a stiffness matrix for a structure represents one degree of freedom of the FE mesh, partitioning by rows corresponds to assigning those DOFs of the FE mesh to a processor. After the system solution [22], the displacements of these DOFs of the structure will be residents of the processor to which they were assigned. So, we can evaluate, for each element, the part of the element stresses that depend on these displacements in parallel using (17). This stress evaluation approach, we believe, is more efficient than first concentrating all the displacements of some elements in one processor because:

- (1) It does not involve cooperation between processors at all.

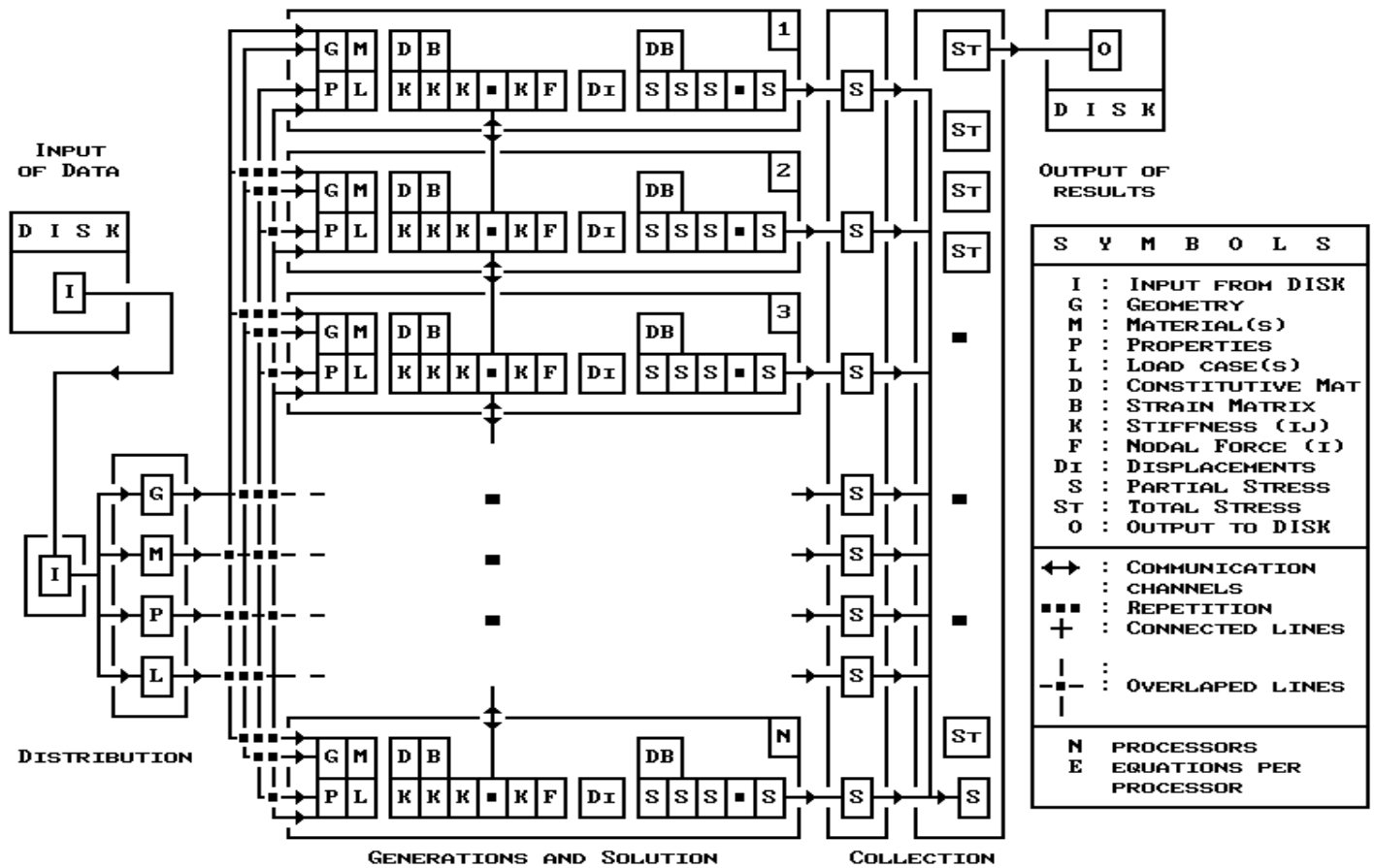


Fig. 10. PARFES.

(2) It can proceed as soon as some displacements have been evaluated in a processor.

However, this parallel approach introduces another stage in the evaluation of the stresses, namely the formation of the total stresses for an element by adding the appropriate partial ones using (14). This extra stage must be applied on all the elements for which stress evaluation depends on displacements which reside in different processors. For elements where all required displacements are at the same processor, an element stress evaluation using (4) can be obtained. It should be mentioned here that if a column by column partitioning and generation scheme of the stiffness matrix and loads vectors is selected, all nodal displacements for one loading or more will be in one processor after the system solution. Communication of the displacements to all other processors must be done in order to evaluate in parallel the element stresses. Therefore a column-wise partitioning will be more expensive than a row-wise one during the stresses evaluation stage.

Before using (15) and (16), or (7) and (8), a transition from serial data to parallel data, and vice versa after using (17), or (4), must be performed. The input data that describe the FE mesh, its properties, and its loadings must be distributed to the appropriate processors so that the parallel computation can proceed. The partial stresses residing in different processors, after their evaluation using (17), have to be collected and added, by using (14), for the evaluation of the total stresses for each element. After that

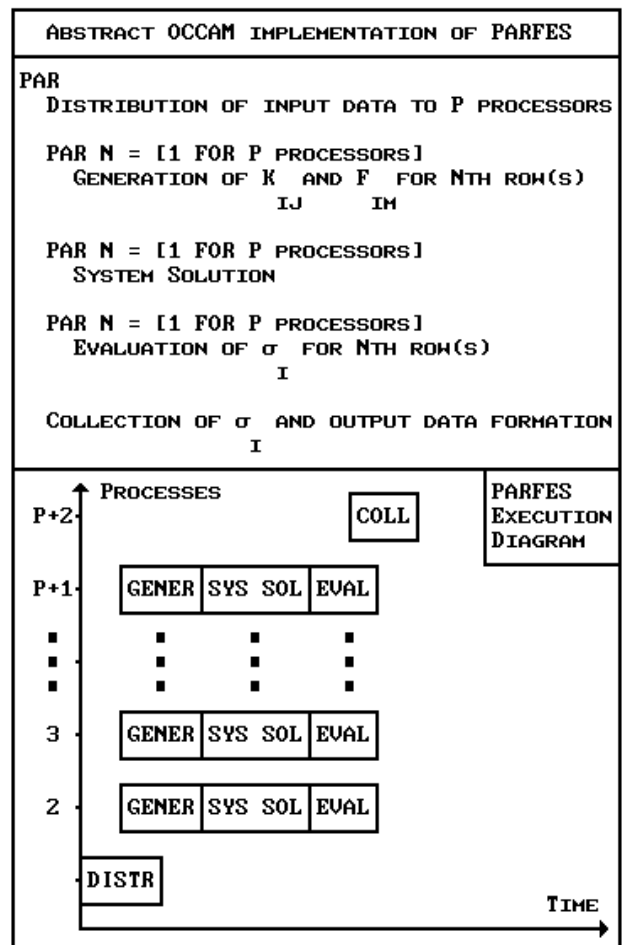


Fig. 11. PARFES in OCCAM and execution diagram.

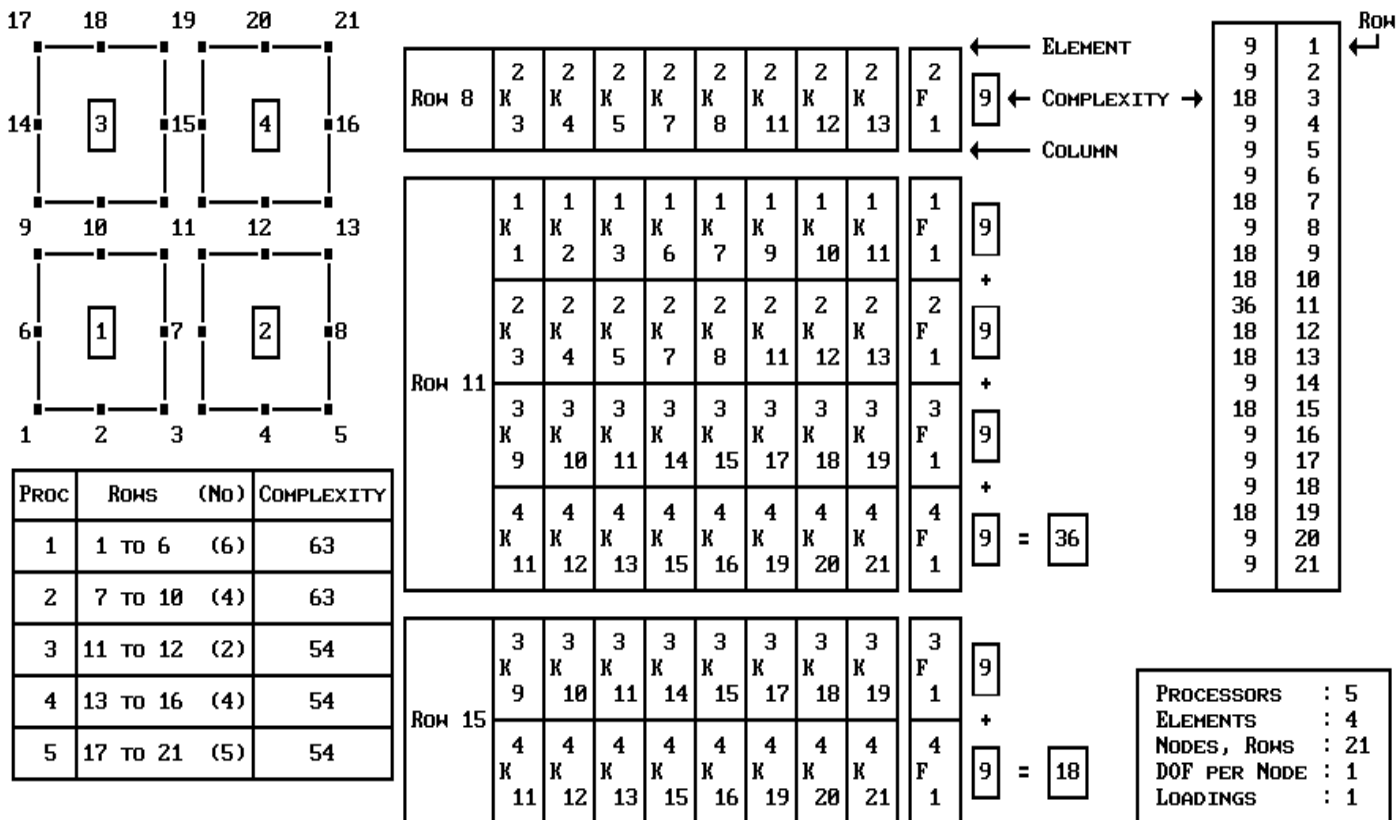


Fig. 12. Partitioning for similar complexity.

an output file with the nodal displacements and the element stresses will be constructed.

A diagram of the proposed Parallel Finite Element System (PARFES) is shown in Fig. 10. A general and abstract outline of the processes that implement PARFES, written in OCCAM, and a representation of its execution are shown in Fig. 11.

IMPLEMENTATION

The OCCAM program that implements the Parallel FE System proposed above is described briefly here. PARFES performs the linear static analysis of structures by the displacement method. For the purpose of this illustration the isoparametric, eight noded plane stress/strain element

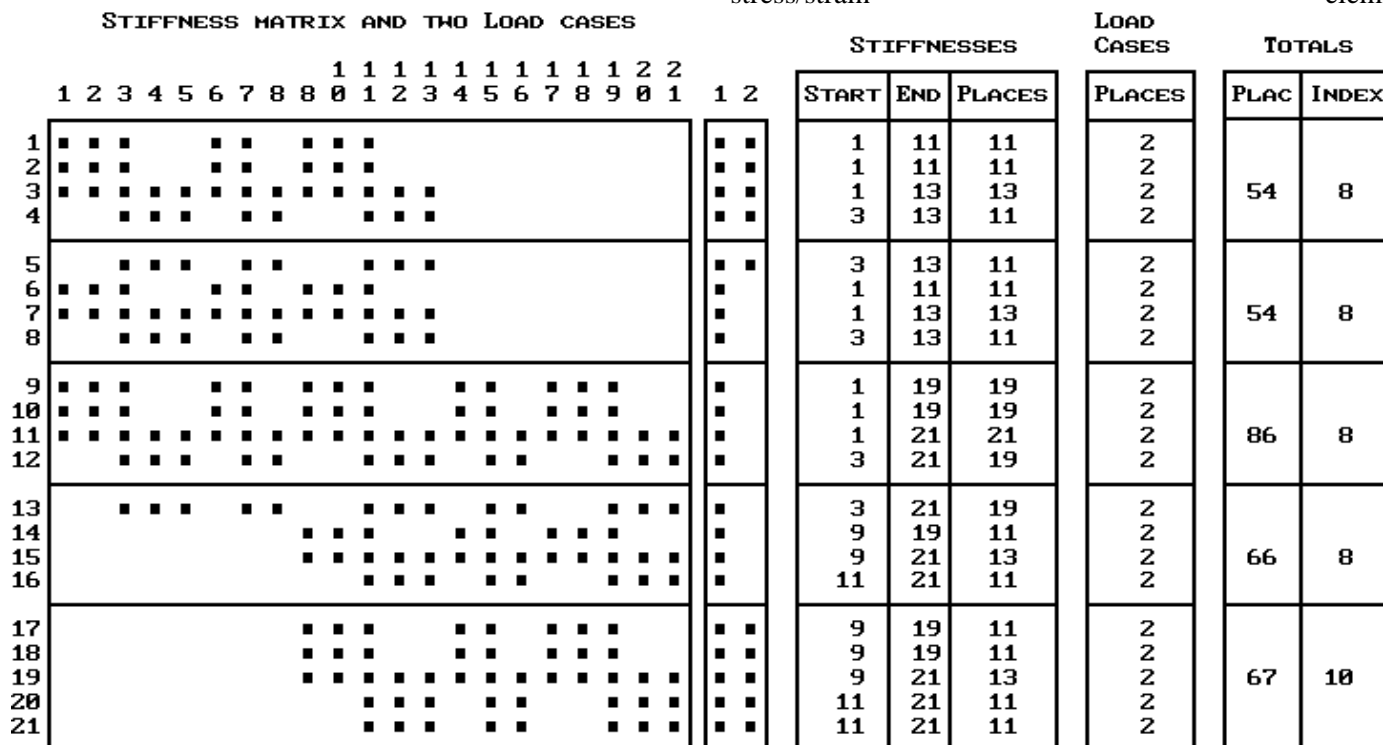


Fig. 13 Storage requirements.

under a uniformly distributed loading pattern [25] will be used as an example. Other types of elements and loadings will not make any difference in the implementation strategy that is outlined here because their treatment will be the same from the parallel algorithm point of view. PARFES takes as input a file with the data that describe the FE mesh and the loadings, and produces an output file with the results.

PARFES consists of three major parts:

(A) Distribution of input data to P processors (serial to parallel).

(B) FE static analysis with displacement method using P processors (parallel).

(C) Collection of output data from P processors (parallel to serial).

In order to obtain maximum utilization of the P processors of a parallel computer, parts (A) and (C) of PARFES have been written. These parts can be executed on the front-end computer used for the parallel computer because their computational needs are minimal compared with part (B). In any case, if (A) and (C) are executed on the parallel computer, we will find that during execution most of the P processors stay idle, a situation that we want to avoid.

Part (A) of PARFES carries out the following steps:

(A1) Read input data from disk to memory.

(A2) Decide how many and which rows (equations) of the structure's stiffness matrix and loadings (RHS) should be created by each processor.

(A3) Decide what solution method for the system should be followed (direct or iterative).

(A4) Send the appropriate data to each processor according to (A2).

While steps (A1) and (A4) are straightforward, steps (A2) and (A3) need some more explanation. Steps (A2) and (A3) depend on:

(I) The number of processors available.

(II) The computational complexity that is involved for the generation of each row of the system.

(III) The computational complexity that is involved in using each row during system solution.

(IV) The communication complexity during system solution.

An example of (I) and (II) is shown in Fig. 12, while (III) and (IV) are examined in more detail in [22].

Part (C) of PARFES carries out the following steps:

(C1) Collect displacements, partial and total stresses from all P processors.

(C2) Form the total stresses which required.

(C3) Write output data from memory to disk.

Part (B) of PARFES starts when each processor has its own input data for the solution of an FEM problem and

ends when output results have been obtained, i.e. displacements, partial and some total stresses.

Part (B) of PARFES carries out the following steps:

(B1) Each processor generates the rows of the structure's stiffness matrix and loadings that have been assigned to it.

(B2) System solution with P processors.

(B3) Each processor holds a part of the total displacements vectors and calculates the partial stresses due to these displacements.

Of the above three steps, (B2) is examined in [22] in more detail. Steps (B1) and (B3) involve several matrix operations according to eqns (15)-(17). The OCCAM processes that form steps (B1) and (B3) have been written in such a way that vectorization techniques can be exploited. This means that the most common vector operations, e.g. vector addition and multiplication, have been written as OCCAM processes and are used throughout PARFES. If a particular parallel computer supports dedicated hardware for vector operations, at the processor level, a faster execution of (B1) and (B3) can be obtained by replacing the above vector processes written in OCCAM with the corresponding ones for the parallel computer before the OCCAM source of PARFES is compiled. Otherwise the vector operations are performed by each processor with the aid of the vector processes written in OCCAM. The replacement must be done manually because current versions of the OCCAM compiler do not support any kind of automatic vectorization. Finally it should be mentioned that the current version of PARFES is designed with the aid of the Transputer Development System [34] executing on the VAX 8600 of the Imperial College Computer Centre.

STORAGE REQUIREMENTS

In order to proceed to the generation of the system of eqns (6), we must have at each processor the appropriate data that are used in (15), (16) and (7), (8). That data are distributed to each processor after the assignment of equations to processors. For one processor, and for the assigned equations only, the data are:

(1) Nodal coordinates.

(2) Element connectivity data.

(3) Element properties and material.

(4) Shape functions for the participating elements.

(5) Data for Gauss points (coordinates and weighting).

During the stiffnesses and loads generation, certain generated data are useful for the stresses evaluation process, e.g. the product of the constitutive matrix and the strain matrix for a Gauss point. For a specific parallel computer one of the two following strategies will be adopted:

(a) Store the intermediate data on mass storage (disk) and recall it when the stresses evaluation process is executed.

(b) Recalculate the intermediate data when they are needed for the stresses evaluation process.

The factor that will permit us to decide which one of the above strategies should be followed in a particular parallel computer is how the available computation speed compares with the speed of storing and recalling data from the disk.

For each particular row that resides in a processor of a parallel computer the following storage scheme is adopted:

(a) K places are reserved for each row starting from the lowest non-zero position to the highest one, inclusively. Two pointers, one pointing to each end, are used so the exact position of the band in the row is known. The values for these pointers can be found from the connectivity data before generation and assembly proceed.

(b) M places are reserved for the M load cases.

In Fig. 13 the requirements of input data and storage for the FE mesh of Fig. 12 are shown for each processor.

CONCLUSION

A portable and adaptable Parallel Finite Element System (PARFES) is proposed. In this paper parallel techniques for element stiffness generation, loads generation and element stresses evaluation are presented. Details are given for the type of parallel computer on which such a system will be used (MIMD), as well as for the parallel computation model and the programming language (CSP and OCCAM) that have been adopted for the creation of PARFES. Implementation details for PARFES are given, such as the organization of the program, and the computation and storage strategies that should be used in accordance with the characteristics of a particular parallel computer. Communication details and strategies are not given because communication between processors is not required at all for the stages of the parallel processing techniques for FE analysis which are considered here. Finally, the expected speedup for these techniques according to (18) is almost P , where P is the number of processors, if the assigned tasks to them are similar, because no co-ordination time is spent at all.

Acknowledgements—CRAY is the registered trademark of Cray Research, Inc. OCCAM and Transputer are registered trademarks of the INMOS Group of companies. VAX and VMS are registered trade marks of the Digital Equipment Co.

REFERENCES

1. A. K. Noor and W. D. Pilkey (Editors), *State-of-the-Art Surveys of Finite Element Technology*. The American Society of Mechanical Engineers, New York (1983).
2. C. A. Brebbia (Editor), *Finite Element Systems: A Handbook*. Springer, New York (1982).
3. J. P. Riganati and R. B. Schneck, Supercomputing. *Computer* **17**(10), 97-113 (1984).
4. D. D. Gajski and J. Peir, Essential issues in multiprocessor systems. *Computer* **18**(6), 9-27 (1985).
5. R. W. Hockney and C. Jesshope, *Parallel Computers*. Adam Hilger, Bristol (1981).
6. M. J. Flynn, Very high speed computing systems. *Proc. IEEE* **54**, 1901-1909 (1966).
7. M. J. Flynn, Some computer organizations and their effectiveness. *IEEE Trans. Comput.* **21**, 948-960 (1972).
8. K. Hwang, *Tutorial, Supercomputers: Design and Applications*. IEEE Computer Society Press, New York (1984).
9. C. Noorie, Supercomputers for superproblems: An architectural introduction. *Computer* **17**(3), 62-75 (1984).
10. W. R. Graham, A review of the capabilities and limitations of parallel and pipeline computers. In *Numerical and Computer Methods in Structural Mechanics* (Edited by S. J. Fenves, N. Perrone, J. Robinson and W. C. Schnobrich), pp. 479-495. Academic Press, New York (1973).
11. J. L. Rogers, The impact of fourth generation computers on NASTRAN. NASA TM X-3278, September 1975, pp. 431-447.
12. A. K. Noor and S. J. Voigt, Hypermatrix scheme for finite element systems on CDC STAR-100 computer. *Comput. Struct.* **5**, 287-296 (1975).
13. A. K. Noor and S. J. Hartley, Evaluation of element stiffness matrices on CDC STAR-100 computer. *Comput. Struct.* **9**, 151-161 (1978).
14. A. K. Noor and J. J. Lambiotte, Finite element dynamic analysis on CDC STAR-100 computer. *Comput. Struct.* **10**, 7-19 (1979).
15. J. C. Knight, The current status of super computers. *Comput. Struct.* **10**, 401-409 (1979).
16. G. A. Strohkorb and A. K. Noor, Potential of minicomputer-array processor system for nonlinear finite element analysis. *Comput. Struct.* **18**, 703-718 (1984).
17. J. A. Swanson, G. R. Cameron and J. C. Haberland, Adapting the ANSYS finite element analysis program to an attached processor. *Computer*. **16**(b), 85-91 (1983).
18. W. K. Brown, NASTRAN on the HEP. Tenth NASTRAN User's Colloquium, NASA CP-2249, November 1982, pp. 187-203.
19. W. K. Brown and P. R. Pamidi, COSMIC/NASTRAN on the Cray computer systems. Twelfth NASTRAN User's Colloquium, NASA CP-2328, August 1984, pp. 47-53.
20. J. F. Gloudeman, The anticipated impact of supercomputers on finite element analysis. *Proc. IEEE* **72**, 80-84 (1984).
21. D. Zois, Parallel processing and finite elements. Internal Report, Aeronautics Dept, Imperial College, London (1985).
22. D. Zois, Parallel processing techniques for FE analysis: system solution. *Comput. Struct.* **28**, 261-274 (1988).
23. C. A. R. Hoare, Communicating sequential processes. *Commun. ACM* **21**, 666-677 (1978).
24. C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ (1985).
25. E. Hinton and D. R. J. Owen, *Finite Element Programming*. Academic Press, New York (1977).

26. O. C. Zienkiewicz, *The Finite Element Method*. McGraw-Hill, New York (1977).
27. C. A. Mead and L. Conway, *Introduction to VLSI Systems*. Addison-Wesley, Reading, MA (1980).
28. C. Wu and T. Feng, *Tutorial: Interconnection Networks for Parallel and Distributed Processing*. IEEE Computer Society Press, New York (1984).
29. C. Seitz, The cosmic cube. *Commun. ACM* **28**, 22-33 (1985).
30. R. H. Kuhn and D. A. Padua, *Tutorial on Parallel Processing*. IEEE Computer Society Press, New York (1981).
31. INMOS Ltd, *OCCAM: Programming Manual*. Prentice-Hall, Englewood Cliffs, NJ (1984).
32. U. Schendel, *Introduction to Numerical Methods for Parallel Computers*. Ellis Horwood, Chichester, U.K. (1984).
33. S. Yalamanchili and J. K. Aggarwal, *Reconfiguration strategies for parallel architectures*. *Computer* **18**(12), 44-61 (1985).
34. INMOS Ltd, *Transputer development system. VAX/VMS Version*, Bristol, U.K. (1985).