NUMERICAL AERODYNAMIC SIMULATION FACILITY

F. R. Bailey and A. W. Hathaway
NASA Ames Research Center

INTRODUCTION

The rapid advancement of computer speed and storage over the last two decades has fostered an equally rapid advancement in computational fluid dynamics simulation capability. A major growth field is computational aerodynamics, which combines the disciplines of aerodynamics, fluid physics, mathematics, and computer science for the purpose of simulating aerodynamic flow fields through the numerical solution of approximating sets of fluid dynamics equations. The field of computational aerodynamics, even in its current formative stage, is emerging as an important aerodynamic research and design tool.

Critical to the advancement of computational aerodynamics capability is the ability to simulate flows about three-dimensional configurations that contain both compressible and viscous effects, including turbulence and flow separation at high Reynolds numbers. While it is currently possible to accomplish this in two dimensions, bridging the gap to three dimensions is beyond the capability of current computers.

The Numerical Aerodynamic Simulation Facility (NASF) is proposed to provide this needed increase in computational capability by carefully matching the characteristics of the problems to be solved with advances in computing system architecture. The NASF Project has been under way for approximately three years, and the remainder of this paper will describe Project activities to date.

CHARACTERISTICS OF NAVIER-STOKES SOLUTION ALGORITHMS

The first step in the development of the NASF was to ascertain whether the problems to be solved were such that architectural innovations could reasonably be expected to produce the performance gains desired. To this end, analyses were conducted of two solution techniques for solving the Reynolds averaged Navier-Stokes equations describing the mean motion of a turbulent flow with certain terms involving the transport of turbulent momentum and energy modeled by auxiliary equations. The first solution technique (ref. 1) is an implicit approximate factorization finite-difference scheme applied to three-dimensional flows. The implicit formulation avoids the restrictive stability conditions when small grid spacing is used to obtain spatial resolution of large gradients in viscous dominated regions. The approximate factorization reduces the solution process to a sequence of three one-dimensional problems with easily inverted matrices. The second technique (ref. 2) is a hybrid explicit/implicit finite-difference scheme which is also factored and applied to three-dimensional flows. In this scheme, implicit techniques are used only in the surface normal direction where the grid spacing is the finest. Both methods are applicable to problems with highly distorted grids and a variety of boundary conditions and turbulence models. These early analyses indicated the following three fundamental characteristics:

1) Massive but rather simple calculations

   While a typical Navier-Stokes solution will involve approximately a trillion calculations, these calculations are almost evenly divided between adds and multiplies, with very few divides (on the order of 3%) and essentially no intrinsic functions such as square root. In addition, almost one-third of all computations are in the form of "multiply-add" combinations.

2) Massive but well structured data bases

The large number of grid points necessary to describe non-trivial geometries gives rise to data bases for Navier-Stokes problems of on the order of 40 million words. In addition, this data must be accessed in each of the three spatial directions associated with the factored algorithm approach at least once per iteration. There is only local grid data interaction, however, allowing the sweeps through the data to take place independently.

3) Simple control

The basic algorithms consist of identical operations performed on large blocks of data, which makes them quite well suited to parallel solution approaches. In addition, there are relatively few recurrences or branches, and those that do exist are within deeply nested loops, again allowing the use of parallelism to improve computational speed.

These results clearly indicated that the Navier-Stokes algorithms were in fact ideally suited to parallel processing techniques, the only architectural alternative for meeting NASF goals in a reasonable time frame.

PRELIMINARY NASF SPECIFICATIONS AND GUIDELINES

In addition to providing advanced capability for computational aerodynamics research, a basic goal for the NASF is to be a tool to aid in the design of aerospace vehicles. To serve such a role, it is necessary that solutions be available in relatively short times to make it practical to sort through many possible configurations early in the design cycle when aerodynamic factors have the largest impact on the shape of a new vehicle. This gives the following basic statement of required NASF performance:

> The solution of the Reynolds-averaged Navier-Stokes equations, for grids of one million points, in less than ten minutes.

Assuming reasonable advances in numerical method efficiency (estimating a factor of four improvement by 1983), this in turn gives the following estimates for processing rate and storage:

> A processing rate of approximately one billion floating point operations per second, for the add/multiply/divide mix of Navier-Stokes algorithms, on a data base of 40 million words.

This number of one billion floating point operations per second has served as a quick-reference performance figure throughout the Project. It should be kept in mind, however, that the basic performance goal of the NASF is <u>not</u> stated in terms of raw processing rate, but rather in terms of elapsed time for Navier-Stokes solutions. This of course has profound implications on total system architecture, benchmarks, and so forth.

In addition to the above performance goal, the NASF has very stringent requirements on reliability and trustworthiness. Reliability in this sense means that the machine must be capable of performing useful work a high percentage of the time -- it must not break very often and when it does, it must be easy to repair -- and trustworthiness means that the computations it does produce must be correct -- the user must not have any reason to distrust results. The NASF will be a very large and very complex system; without early and strong emphasis on both reliability and trustworthiness its value for practical engineering use can be severely compromised.

In order to assure the attainment of these goals, one other concept is being strongly emphasized throughout the project: the NASF is <u>not</u> being developed as a computer science project, but rather as the construction of a fluid dynamics research and engineering tool. Therefore, while state of the art technologies must be used throughout in order to reach the substantial performance expectations, there are no specific attempts being made to <u>stretch</u>

the state of the art in individual areas. For example, a basic concept of the Facility views it as being made up of two separate components:

Flow Model Processor (FMP)

> The FMP is the computational engine and storage for solving the Navier-Stokes problems. As it will operate on only one job at a time, it will also contain sufficient staging memory to allow buffering of output for the current and previous job as well as input of code and data for the next job.

Support Processor System (SPS)

> The SPS will be responsible for the overall operation of the Facility, including compiling and scheduling jobs for the FMP, managing the file system, and providing an interactive user interface.

In this concept, then, essentially all custom hardware will be in the FMP itself; the SPS will be a standard, proven, off-the-shelf computing system. Likewise the development of custom software will be minimized: the FMP will have the minimum operating system possible for staging of input and output, and the SPS will have minimum modifications to its standard operating system. This philosophy is also being followed in the development of the FMP programming language. First, it will be based on ANSI FORTRAN 77, with only the minimum extensions required for expressing problem parallelism and memory hierarchy management. Second, there will be no attempt made to have the FMP compiler automatically recognize parallelism in standard FORTRAN constructs; all problem parallelism must be explicitly specified by the programmer. It is strongly felt that both of these restrictions are necessary in order to develop, in the time frame necessary, an operational compiler capable of delivering acceptable FMP performance.

Thus we see that while the NASF has substantial goals in terms of performance and reliability, every effort is also being taken to ensure that it is actually buildable and usable. The Facility is not based on any breakthroughs in either hardware or software technology and will instead achieve its goals through architectural innovations and through the combining of several state of the art technologies under strict reliability guidelines and controls.

PRELIMINARY FEASIBILITY STUDIES

After the encouraging results of the early Navier-Stokes analyses, two independent, parallel feasibility studies were conducted, one by Burroughs Corporation and one by Control Data Corporation. The major efforts of these contracts were the analysis of the two Navier-Stokes algorithms described above and the development of candidate processor architectures optimized for their execution. Extensions of these studies further refined the two baseline configurations, including the development of simulators for performance and functional verification. These two baseline configurations are described later.

A second phase of feasibility studies is currently under way, with the same contractors, with the goal of extending the baseline designs even further, including software and total facility specifications. These further studies will also assess the proposed configurations for weather/climate simulation and will investigate modifications which could improve FMP performance for such applications.

In addition to these two studies, Ames is involved in numerous other contract efforts. Most of these efforts are for technical assistance in areas such as evaluation of feasibility study results, reliability considerations, and performance analysis.

Before discussing the baseline architectures in detail, it is necessary to review two fundamental concepts.

FUNDAMENTAL ARCHITECTURAL CONCEPTS

The first of these concepts is that of parallelism, as illustrated in figure 1. As indicated earlier, Navier-Stokes solutions involve many operations being done repetitively on large blocks of data points; this is illustrated by the program segment in figure 1a. This loop will cause the three operations indicated by the boxes to be done N times, once for I=1, once for I=2, and so forth; each execution of the three operations is called an "instance." Further it is assumed that the N instances are data independent, that results of one instance are not used as inputs by another. (This condition is generally satisfied by the types of problems envisioned for the NASF.) Doing the N independent instances of the loop sequentially, as on current general purpose computers, is illustrated in figure 1b. Since this cannot hope to achieve the processing rate planned for the NASF, some form of parallelism must be used; this is illustrated in figure 1c.

Given this general form of parallelism, it is then necessary to map the N instances of the loop onto physical hardware; methods for doing this are shown in figure 2. One technique, known as "horizontal slicing," consists of performing all N instances of the first operation in the loop, then performing all N instances of the second operation, and so forth. This is the technique used in vector or pipeline computers such as the STAR-100, the ASC, and the CRAY-1. The second general technique, known as "vertical slicing," consists of assigning the N independent instances to N independent processors and having them each perform all of the operations for one particular instance. This technique is commonly referred to as parallel or concurrent processing.

Both forms of slicing assume that in fact N operations can be done in parallel, and unfortunately this assumption is frequently not valid on real hardware. In this case, the problem is solved as a combination of serial and parallel computations. First assume a vertical slicing approach, with P processors available (P<N). The N instances are grouped into M "cycles," with P instances per cycle. M is of course chosen so that MxP is equal to or slightly larger than N. The P processors are then started computing the instances in the first cycle. When they complete, the second cycle is done, and so forth until all M cycles are completed (with some processors possibly being idle during the last cycle). This same problem can manifest itself in the horizontal slicing approach as well if the number of data elements which can be operated on at once is less than N. This can be caused either by exceeding the maximum allowable vector length (as on the CRAY-1 if N>64) or by having insufficient memory to store long vectors of temporary results (as frequently happens on the STAR-100). Again the solution is to perform a series of parallel operations.

Figure 3 illustrates a slightly different problem, more typical of applications other than Navier-Stokes. As shown in figure 3a, the N independent instances in this case involve data dependent or subscript dependent branching ("subscript dependent" means things like boundary conditions). In the horizontal slicing approach, shown in figure 3b, the steps are as follows: perform the first operation for all N instances, make the decision for all N instances (storing the result as a logical or bit vector), perform the third step for all applicable instances (under control of the logical vector produced earlier), perform the alternate third step for the other instances (again controlled by the logical vector), and finally perform the fourth step for all. Actual implementations of "under control of the logical vector" may involve techniques such as sparse vector processing, "gather" operations, index lists, and so forth. Additional overhead is required, however, and all operations must take place in rigid sequence. In the vertical slicing approach each of the individual processors is free to take only the appropriate branch; there is no need to wait on the alternate not selected. This means that the net time for completing all instances can be significantly reduced. Note also that in this type of problem the horizontal slicing approach is illustrative of both vector processing (STAR-100, CRAY-1, etc.) and lockstep parallel processing (ILLIAC IV, BSP, etc.); the vertical slicing approach is applicable only if the processors are able to execute independently.

The other basic concept relates to storage and accessing of three-dimensional data, as required by the 3-D Navier-Stokes codes. As shown in figure 4, sweeping through the data in each of the three directions associated with the factored algorithms involves picking up planes in each of the three orientations. That is, when advancing in the X direction (having the I subscript advance once per loop), it is necessary to operate on Y-Z planes as vectors (the N instances are composed of all J and K values for the current value of I). Similarly, the Y direction sweep fetches X-Z planes and the Z direction sweep fetches X-Y planes. Since computer memory is not oriented in three dimensions, some mapping must be applied between the high level language construct of a three-dimensional array and the linear memory of a computer; an example based on FORTRAN is given in figure 5. Here we can see that accessing the data in all three directions involves not only fetching contiguous words (the most efficient in most architectures), but also groups of words and individual words separated by constant intervals. Thus the requirement that the FMP permit efficient three-dimensional access to the entire 40 million word data base has significant impact on the FMP architecture.

Note finally that the orderly data accessing of the Navier-Stokes algorithms, together with the local grid data interaction, allows the use of a multi-level memory hierarchy. That is, while it is necessary to have extremely fast access to data being fed directly to processors, other data (previous or subsequent planes) may be contained in a slower, block-accessed backing store, to be moved into the faster processor memory only as required.

We will now describe the baseline NASF configurations developed by the two contractors. Both concepts are still in a state of development and should be considered preliminary and subject to modifications as indicated by further analyses. Nevertheless, they indicate the basic directions being pursued.

CONTROL DATA BASELINE CONFIGURATION

Figure 6 shows a simplified block diagram of the Control Data FMP concept. The basic philosophy espoused by CDC was to build the absolutely fastest pipeline processing unit possible, and then combine as many of them as required to meet the processing rate requirements of the NASF. As might be expected, this gave rise to an architecture which is quite similar to the STAR-100: there is a scalar processing unit, which also contains a control unit to do instruction processing; a lockstep group of eight vector pipeline processing units, with buffer registers; a memory mapping unit, allowing complex memory accessing modes at nearly full vector speed; a main random access memory of eight million words; and a block-accessed CCD (charge coupled device) memory of 256 million words. This CCD memory will serve both as the staging memory (for buffering FMP input and output) and as the primary storage for grid data, with appropriate blocks being moved into the eight million word main memory for actual processing. The architecture also provides for both 64-bit and 32-bit processing modes, with 32-bit processing taking place at twice the 64-bit rate.

Reliability features of the CDC architecture are substantial. All memory contains single-error-correction-double-error-detection (SECDED) circuitry, with appropriate block error correcting codes in the large CCD memory. In addition, the vector processors employ a unique variable-redundancy concept which provides substantial error detection. Each processor actually consists of two identical, independent sets of functional units, plus an input operand select unit and a set of coincidence checkers. For simple operations, such as AxB, the same operand streams are sent to each set of functional units and the units perform identical, redundant calculations. The outputs of each pair of redundant components are sent to the coincidence checkers, providing complete redundancy and a very high degree of error checking. As mentioned above, however, many computations in Navier-Stokes problems are of more complex forms, such as (A+B)x(C+D). The vector processors are also capable of performing such complex calculations although in this case all components would be needed in the actual computation, giving no redundancy and no checking. Operations of intermediate complexity also exist, which allow some

units to operate redundantly and be checked while others operate independently. Thus there is a dynamic trade-off being made between maximum speed, in which all units operate independently, performing three floating point operations every machine cycle, and maximum error detection, with all units performing redundant, checked calculations. It is felt that a normal instruction mix will provide ample checking while still allowing the peak performance rates required of the NASF.

In addition to the error detection provided within the pipelines, there is also redundancy in the form of a ninth, spare processing unit which can be activated at any time. Thus if a processor fails, the spare processor can be configured in and the FMP can resume operation while the failed processor is being repaired. Note that no instruction re-try is planned, however; if a processor fails, the currently active job is aborted prior to reconfiguring the pipelines and resuming operation.

The scalar unit of the Control Data FMP is very similar to the scalar unit of the STAR-100A. This is expected both to save design effort and to produce a more reliable device.

The map unit is a substantial extension of the current STAR-100 architecture in that it provides the capability of accessing memory in each of the three directions shown in figure 5: contiguously (Z sweep), evenly spaced groups (Y sweep), or evenly spaced words (X sweep). There can be a significant performance degradation for non-contiguous accesses, but the map unit operates concurrently with many vector operations, allowing complex fetching of the next vector to overlap with processing of the current one.

Since the Control Data approach is basically horizontal slicing, as is the STAR-100, the proposed FMP programming language draws heavily from STAR FORTRAN. Unfortunately STAR FORTRAN is hardly an ideal high level language due to the need for frequent use of the "Q8" construct, which is essentially embedded assembly language. CDC does feel that they have learned quite a bit from the STAR FORTRAN experience, however, and that reasonable FORTRAN syntactic extensions can in fact be devised which will allow true high level specification of horizontal slicing parallelism. An early attempt had also been made to specify a vertical slicing extension, using a construct which was similar to the Burroughs proposal (to be described later). It was felt that this would require considerably more compiler sophistication than was reasonable, however, and so the approach was dropped.

BURROUGHS BASELINE CONFIGURATION

In the Burroughs NASF baseline configuration, the FMP-SPS interconnection is quite similar to that used for the Burroughs Scientific Processor (BSP) to build on experience and save development costs. The Burroughs FMP design is not at all similar to the BSP, however; figure 7 shows a simplified diagram of the proposed FMP.

The approach taken by Burroughs is also quite different from that of Control Data. Here the processors are relatively simple devices, but the replication factor is much higher. There are 512 independent processors proposed, each with 32,768 48-bit words of local storage for program and data. There is also a control unit, which controls overall FMP operation and allows synchronization of the processors. The control unit does not do instruction decoding and sequencing for the processors, however; the FMP is not a lockstep machine like the ILLIAC IV or the BSP. Instead, each processor has its own program storage and instruction decoding capability. While this potentially allows the FMP to operate as a full multiple instruction multiple data stream (MIMD) machine, this is not the mode of operation planned for the NASF. The primary reasons for this are the primitive interprocessor cooperation capability provided (there is no automatic "locking" of data in extended memory, no capability for synchronization of subsets of processors, etc.) and the lack of a sophisticated program distribution mechanism (it is planned that the same program will be broadcast to all processors at once by the control unit). Therefore the Burroughs FMP is envisioned as a "synchronizable array machine," in which the processors will in fact all

execute the same code with frequent synchronization although individual processors will be able to take different data dependent branches and also to utilize data dependent arithmetic algorithms as well as individual instruction retry and other error recovery procedures.

The memory associated with each processor will in actuality be treated as scratchpad storage, with problem grid data being passed through the processor memories as planes appropriate to the current sweep direction. The actual problem data base will be held in a 34 million word random access memory, which transfers data to and from the processor memories through a transposition network (to be described later). This 34 million word extended memory is organized as 521 separate modules, the prime number 521 being chosen to minimize bank conflicts. That is, with a prime number of memory banks, the only conflicts which will arise are for data intervals of exact multiples of 521. With other numbers of banks, say 512, conflicts could occur for intervals of 2, 4, 8, 16, and so forth.

The Burroughs configuration also utilizes CCD technology for the staging memory, with a proposed 134 million words. Transfers to and from the 34 million word extended memory would be on a job basis, rather than the plane-at-a-time access planned by Control Data.

The Burroughs design also makes use of SECDED in all memories, but error detection within the processors poses a quite different problem than in the fewer pipeline processors of the CDC design. Studies are currently under way to determine appropriate error detection mechanisms for such a large array of processors.

The transposition network originally proposed by Burroughs has undergone some fundamental changes recently, as follows. The original FMP design was tailored specifically to Navier-Stokes problems, and as mentioned earlier, these problems have well structured data accesses and relatively few branches. Thus a network was designed which would provide conflict-free (that is, full speed) accessing for words separated by a constant skip distance. Such a sequence of words located at a constant interval of p words is called a "p-ordered vector." Thus the network would handle the X and Z sweeps at full speed, and sufficiently large arrays would also be efficient in the Y sweep (if the number of words in each group is large compared to the number of processors). This network was quite simple and easy to control, but it did require that all processors access data from different extended memory modules (which will always be the case with p-ordered vectors, due to the prime number of banks) and that all processors be synchronized prior to extended memory accessing.

Subsequent feasibility studies were concerned with the suitability of the FMP for other problems, however, notably weather/climate simulation codes. These programs have significantly more data dependent branching, as well as less orderly data accessing. In fact, analyses showed that significant performance degradations were being caused by the need for synchronization and the restricted extended memory accessing permitted. To solve this problem, a new transposition network is currently under investigation. This network is much more general than the previous one, with processors making requests independently and the network itself determining what path conflicts exist and temporarily blocking conflicting requests. The basic interconnection of the network is similar to an omega network, but the design calls for a self-setting mode of control. That is, rather than having the control unit compute the proper setup for the network (a computation which can guarantee optimum operation but which is prohibitively complex), the network is set up by combinatorial logic cascading through the individual stages. Once a path is established between a processor and an extended memory module, that path is held until all data are passed. The path is then released, allowing blocked requests to proceed.

Simulations of various mechanisms for implementing the new network are in progress, and preliminary indications are that a self-setting network can in fact be built which is as efficient as the old network for contiguous words, only slightly less efficient for evenly spaced single words, and significantly more efficient for spaced groups, particularly small groups. Thus the new

network may cause a small degradation for Navier-Stokes problems (although even this is not obvious since the network accounts for relatively little of the Navier-Stokes time) but should be substantially better for other applications with more random data accessing characteristics.

The Burroughs FMP is basically a vertical sliced architecture: the processors are independent during execution of a cycle of instances, with synchronization necessary only at top and bottom. As such, the programming language extensions are minimal and quite easy to use. Currently the major extension is a DOALL construct, which performs essentially like a standard FORTRAN DO-loop except that the programmer has promised that all instances of the loop are data independent, and thus may all be done in parallel. The compiler has a relatively easy job of carving up instances into cycles, and most serial optimization techniques are applicable.

PERFORMANCE EVALUATIONS

As mentioned earlier, Ames is undertaking independent performance evaluations of each of the baseline FMP configurations. This is being done by first programming each of the codes in the appropriate FMP FORTRAN, then hand compiling them (assuming an unsophisticated compiler), and finally timing the machine code, either by hand or with simulators.

The results of these evaluations, together with maximum processing rates quoted by Control Data and Burroughs, are shown in figure 8. All performance numbers are in terms of billions of floating point operations per second; note that the initial performance specification for the FMP was 1.0.

The first line of figure 8 gives the maximum rates of the two configurations for simple operations (such as AxB). Note that the Control Data configuration performs twice as fast in 32-bit mode. The second line shows the absolute maximum processing rate possible for each architecture, using the most complex operations available (AxB+CxD for Control Data and AxB+C for Burroughs). The next two lines are the performances estimates for the two Navier-Stokes codes provided by NASA; the first is a hybrid explicit-implicit method and the second is totally implicit. Notice that the Control Data figures are for 32-bit; it is unknown how much of the actual code could be executed with sufficient accuracy in this mode. The final two lines are estimated figures for the GISS global circulation model and the MIT spectral model, the two weather/climate codes being considered.

SUMMARY

Although feasibility studies are continuing, it has been fairly well demonstrated that the NASF can in fact be built, that it can meet Navier-Stokes performance objectives, and that it will perform efficiently on other applications as well. Subsequent phases of the Project will verify these conclusions through detailed design, actual fabrication, and subsequent operation of the Numerical Aerodynamic Simulation Facility.

REFERENCES

1. Pulliam, T. H. and Steger, J. L.: On Implicit Finite-Difference Simulations of Three Dimensional Flow. AIAA Paper 78-10, 1978.

2. Hung, C. M. and MacCormack, R. W.: Numerical Solution of Supersonic Laminar Flow Over a Three-Dimensional Compression Corner. AIAA Paper 77-694, 1977.

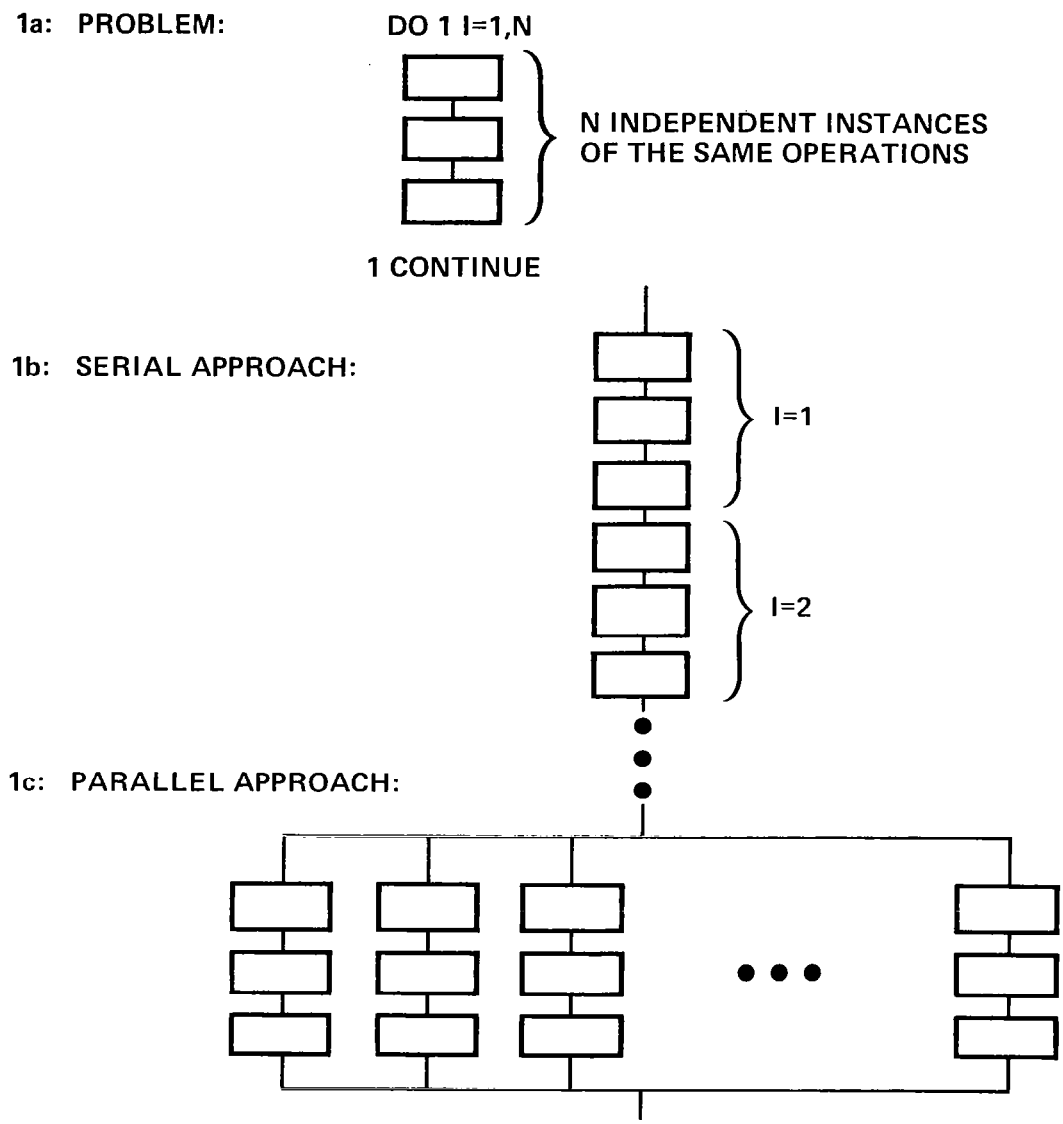
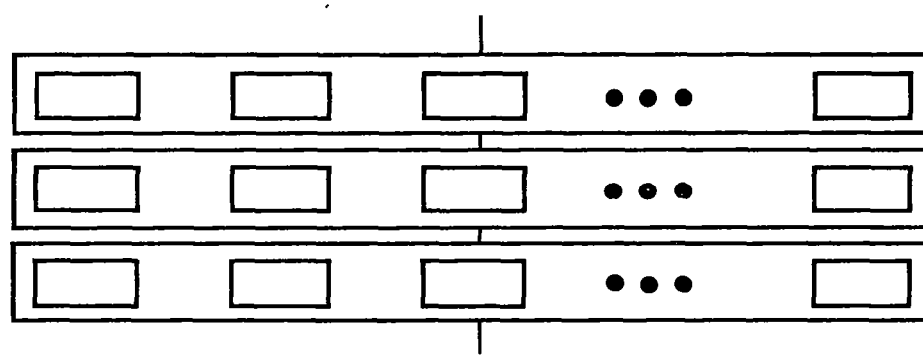


Figure 1.- Concepts of parallelism - basic.

2a: HORIZONTAL SLICING (VECTOR OR PIPELINE APPROACH)



2b: VERTICAL SLICING (MULTIPLE PROCESSOR APPROACH)

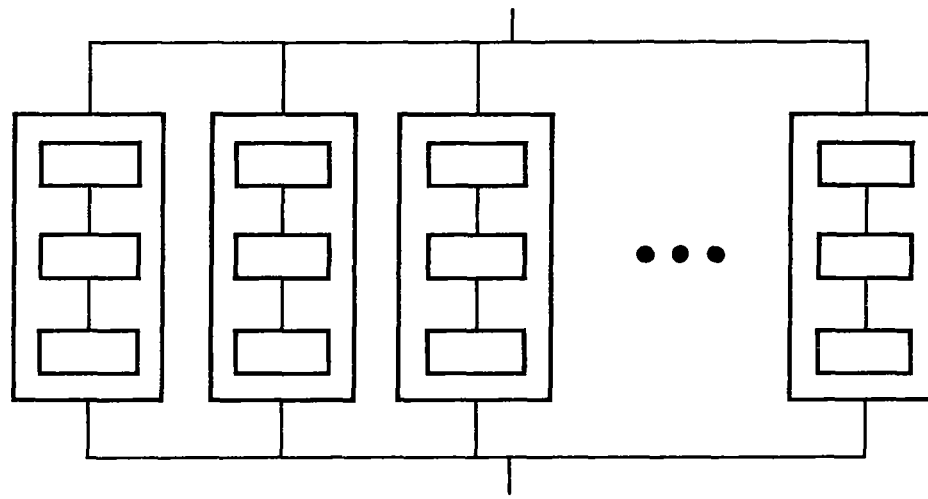


Figure 2.- Concepts of parallelism - slicing.

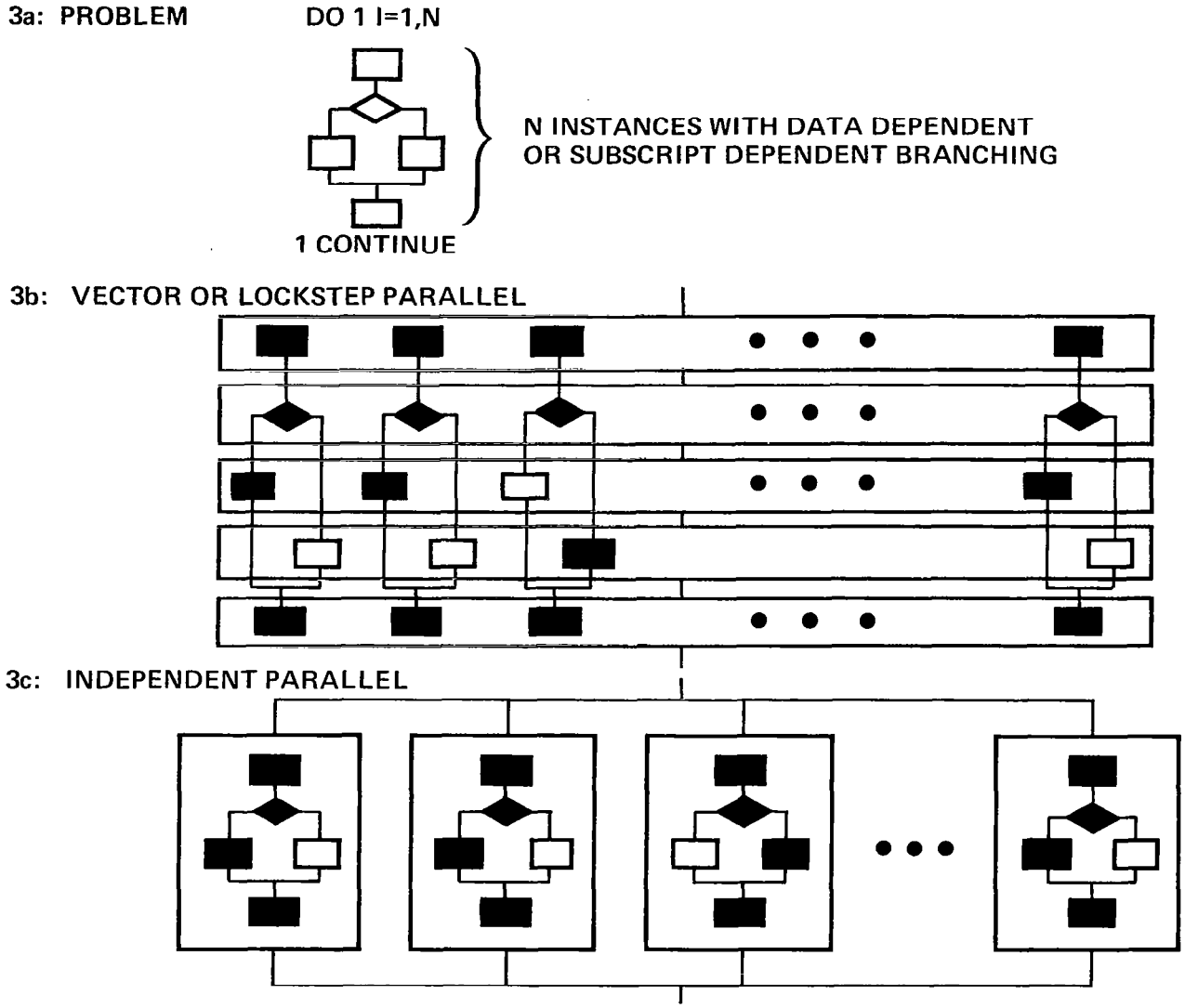


Figure 3.- Concepts of parallelism - branching.

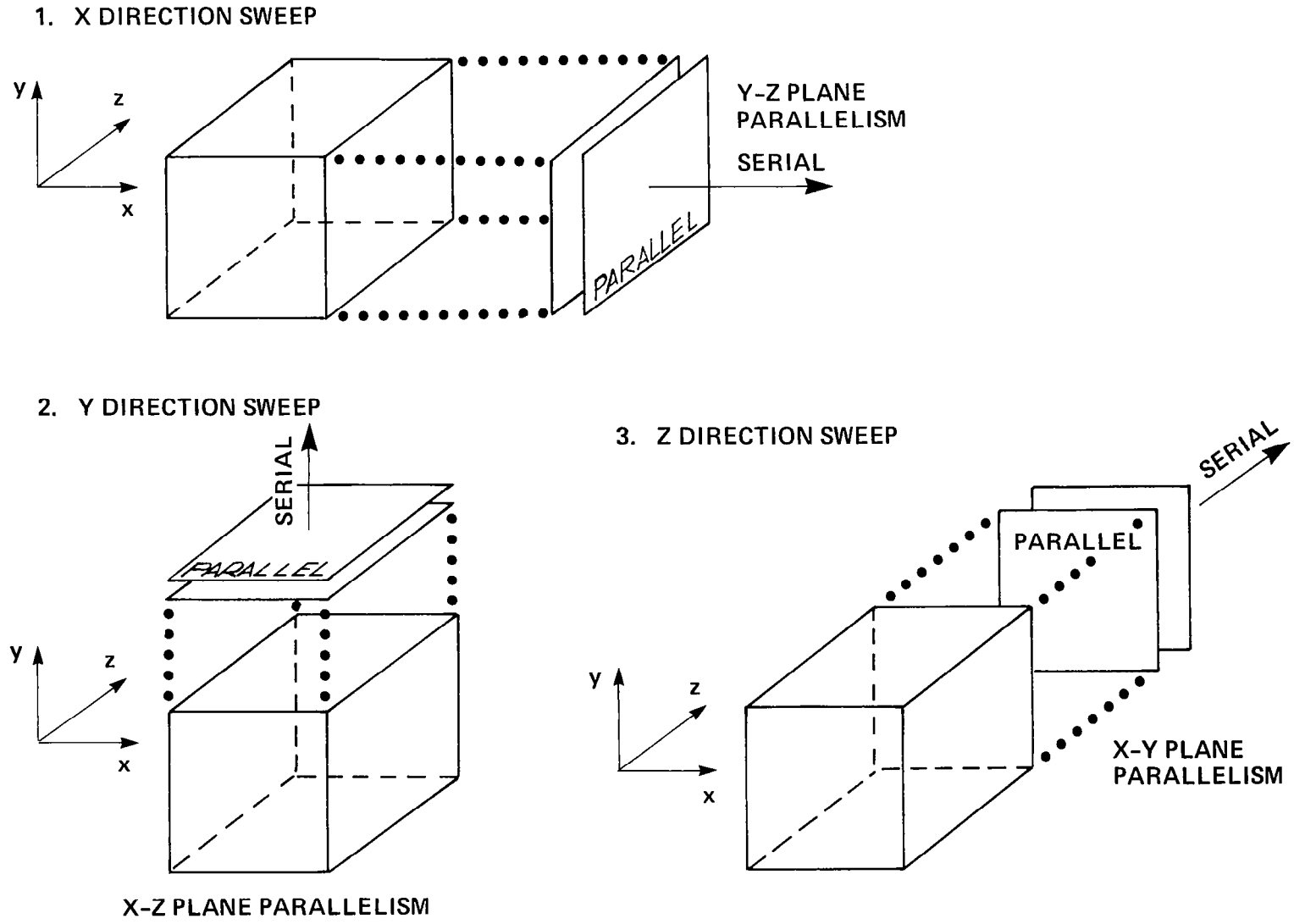


Figure 4.- The three sweeps of the three-dimensional implicit method.

DIMENSION Q(3,3,3)

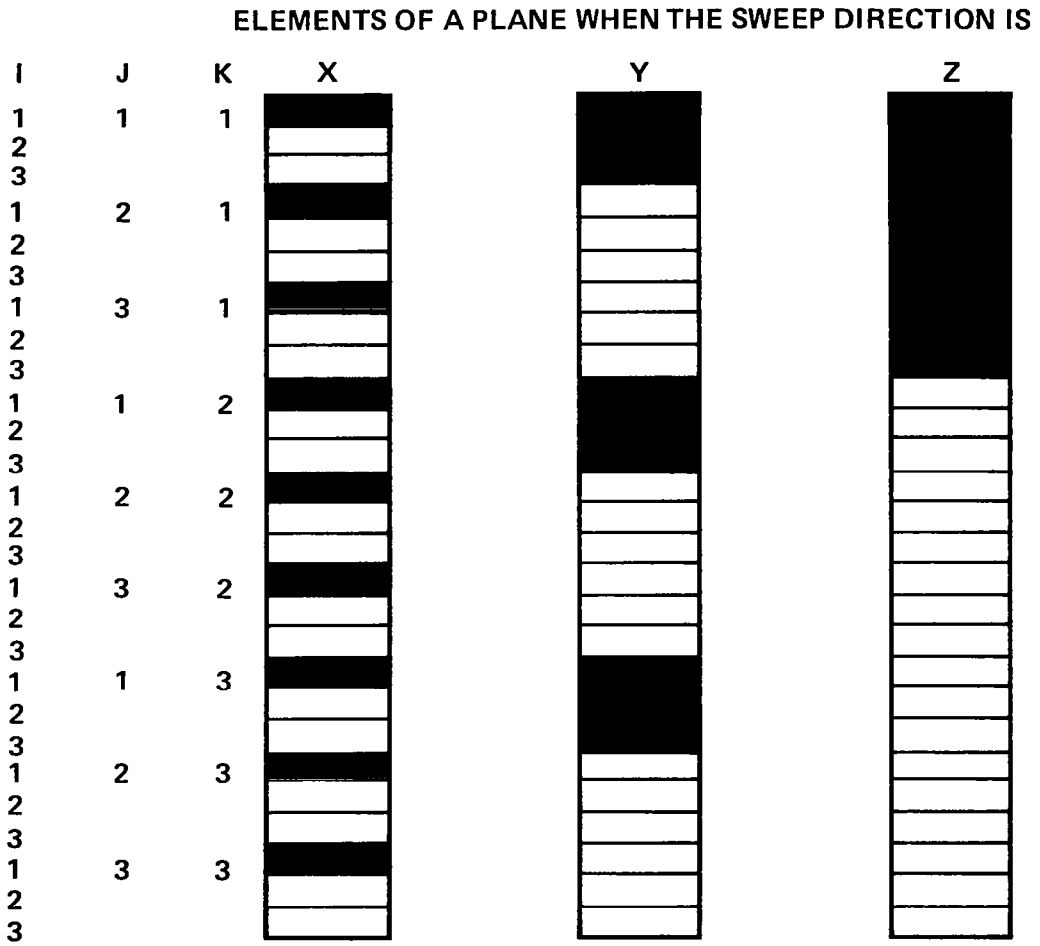


Figure 5. Portrait three dimensional array accessing.

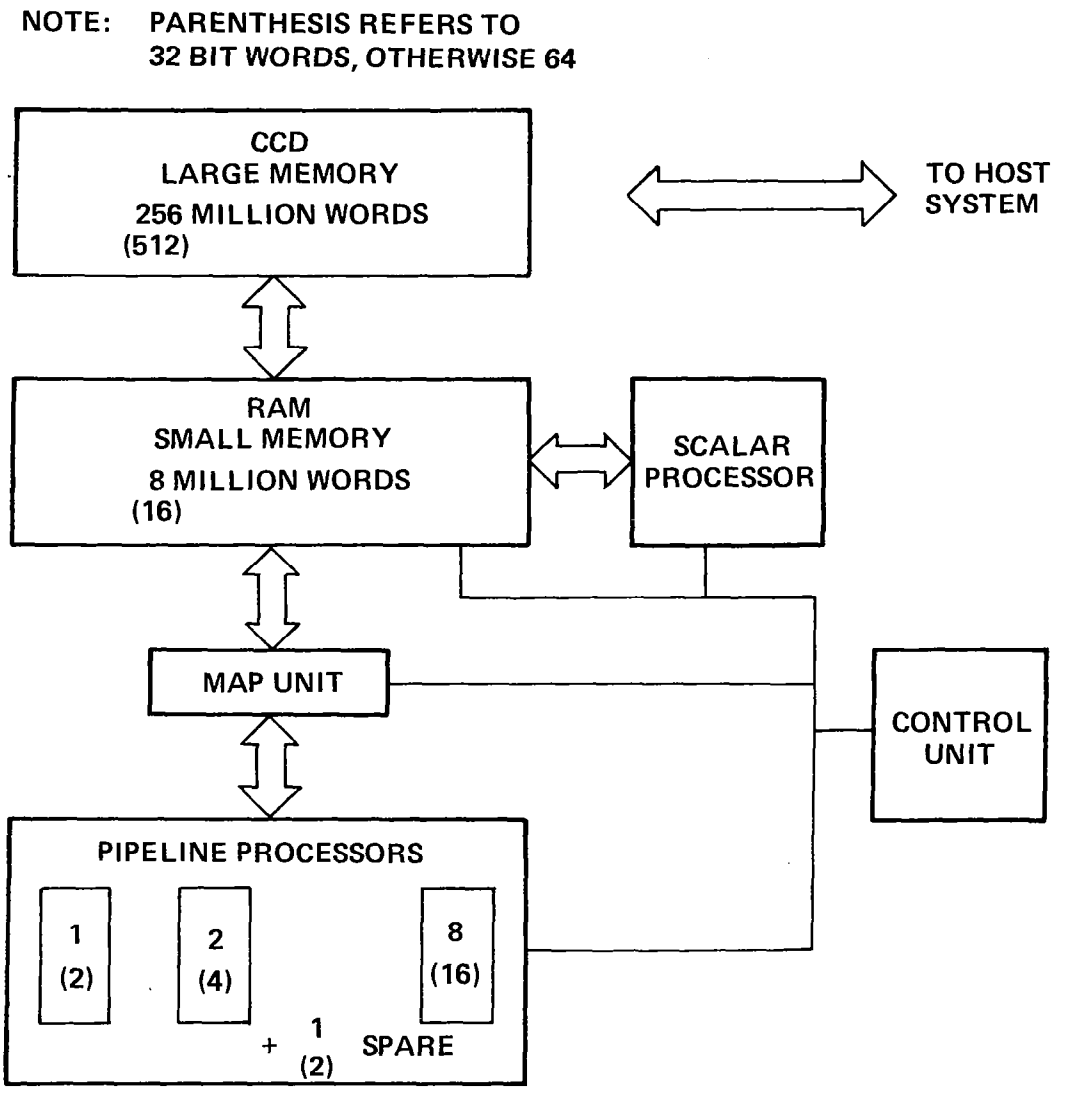


Figure 6.- Simplified block diagram of Control Data FMP.
(RAM denotes random access memory.)

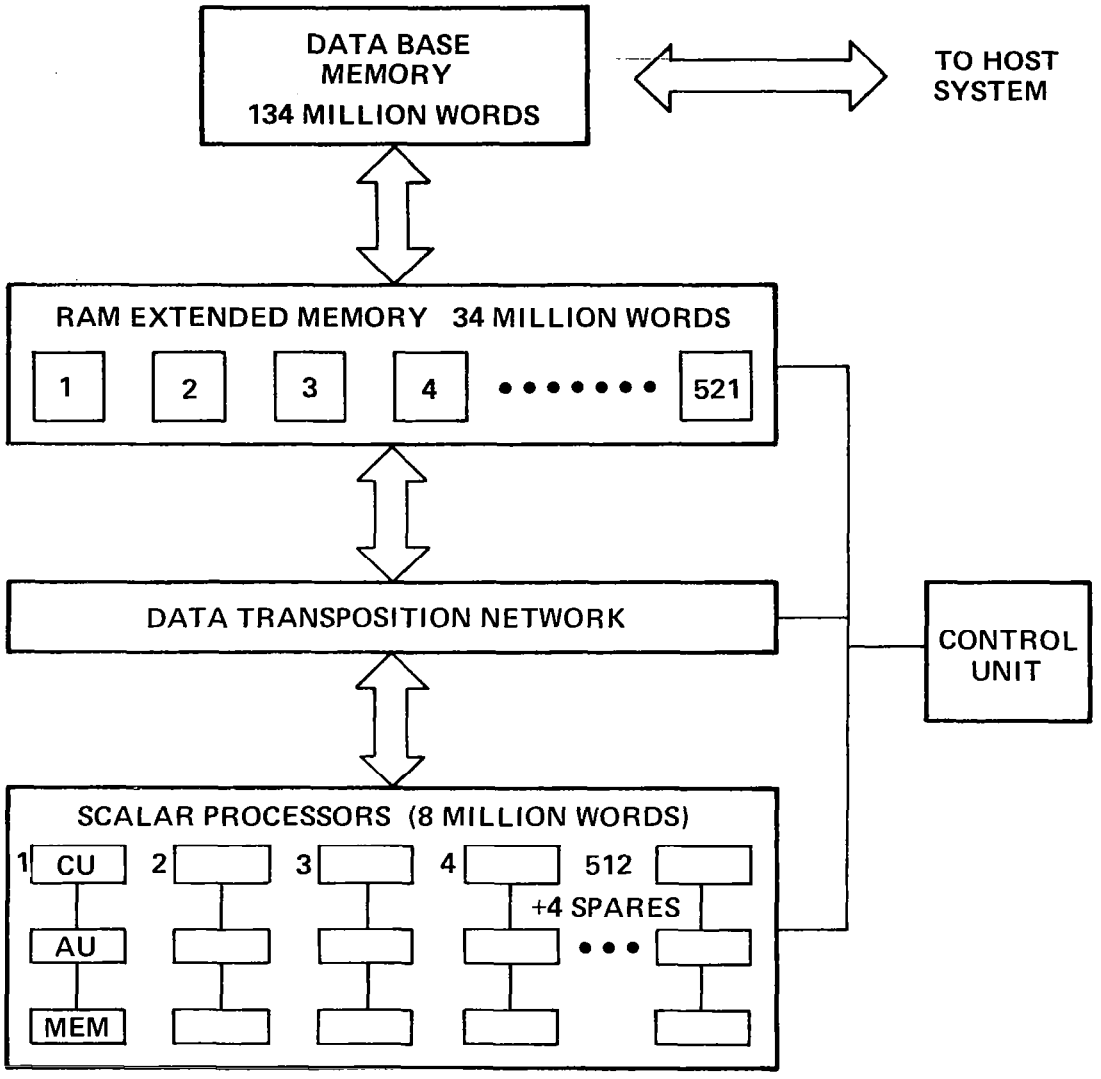


Figure 7.- Simplified block diagram of Burroughs FMP.

IN BILLIONS OF FLOATING POINT OPERATIONS PER SECOND

CODE	CONTROL DATA	BURROUGHS
MAX RATE (SIMPLE)	0.5 (AxB, 64–bit) 1.0 (AxB, 32–bit)	1.5 (AxB)
MAX RATE (COMPLEX)	1.5 (AxB+CxD, 64–bit) 3.0 (AxB+CxD, 32–bit)	2.3 (AxB+C)
HUNG-Mac CORMACK	0.9 (32–bit)	1.1
PULLIAM-STEGER	1.2 (32–bit)	1.2
GISS GCM	0.54 (32–bit)	0.8
MIT SPECTRAL	0.5 (32–bit)	0.7

Figure 8.- Performance estimates for alternate FMP concepts.