Engineering and scientific processing on the IBM 3090

by D. H. Gibson
D. W. Rain
H. F. Walsh

The IBM 3090 processor implementation of the System/370 Vector Architecture represents a major new system design for engineering and scientific processing, featuring both scalar and vector capability in a uniprocessor and in a dyadic and four-way parallel processing environment. The history of large-scale scientific processing is reviewed, leading to a statement of current requirements. The design objectives for scalar, parallel, and vector capabilities are identified, followed by a summary of the resulting 3090 features. Selected highlights of the vector hardware are given, followed by a summary of the supporting software. The paper concludes with a discussion of performance, beginning with the identification of suitable applications. An example is given of one application utilizing each of the three capabilities: scalar, parallel, and vector. Several of the most important performance parameters are identified.

The design of computers well suited to the computational requirements of large-scale engineering and scientific applications can proceed along three complementary lines of implementation. The IBM 3090 processor implementation with the System/370 Vector Architecture combines all three lines—fast scalar processing, parallel processing, and vector processing[1]—in a single product. The problem space addressed by each is depicted in Figure 1. Each of the three must achieve the minimal requirement of providing fast floating-point arithmetic and large memory capacity.

This paper discusses the 3090 Central Electronic Complex design, focusing on the implementation structures that contribute the most to performance and capacity on engineering and scientific applications. The general computational characteristics of these applications are described, showing the need for the fast floating-point execution and large memory, and the opportunity for scalar, parallel, and vector designs. A brief historical review summarizes a few of the preceding fast scalar, parallel, and vector offerings, and then the requirements for large-scale scientific processing in the 1985 – 1990 time frame are discussed. The remainder of the paper follows each of the three complementary capabilities—scalar, parallel, and vector—from design objectives to design features to selected highlights. The paper concludes with examples of measured performance utilizing each of the three capabilities.

General description of engineering/scientific applications

High-performance scientific systems have historically been aimed at the solution of specific appli-

cations in the areas of weather and hydrodynamics. Interest in the use of computers, rather than physical models, is now increasing for fundamental engineering and scientific studies. In one case, a computer costing tens of millions of dollars, rather than a wind tunnel costing even more, has been proposed for studies by NASA Ames. In another, the use of computer simulation runs costing thousands of dollars, rather than construction of a prototype car costing hundreds of thousands, is the preferred method for Detroit automobile manufacturers to conduct structural analysis studies.

The effect of the historical improvement in performance and price-performance of computing systems, together with the increased cost of personnel and the greater availability of prepackaged programs, has resulted in a pronounced growth in the use of computers for engineering and scientific applications. Many designers are relying on the computer throughout the entire development cycle. The use of physical models and "testing to destruction" is being replaced by extensive computer simulation. The consequences are several: shortened design cycles, reduced development cost, improved products, and the ability to attack problems that cannot be solved by any other means.

The well-known computational characteristics of engineering/scientific (E/S) applications may be summarized as follows:

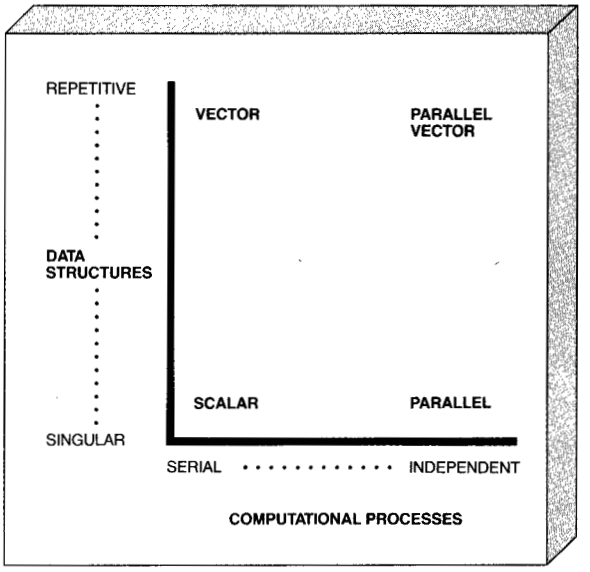
- FORTRAN is the most commonly used language.
- 64-bit floating-point arithmetic is preferred.
- Millions of words of data are often required.
- The trend is towards larger problems, requiring even more memory and greater performance.

In addition, as described below,

- The basic equations affect the computational process.
- The numerical analysis technique used to approximate the basic equations determines the data structures.

These two characteristics, when depicted as in Figure 1, suggest that there is value in different kinds of computers. At any given point in time, there is a limit on the speed of a scalar processor. To provide higher levels of computational capability, it is appropriate to consider computer designs employing concurrency. When the basic equations include independent variables, it is often possible to execute independent processes in parallel. A computer providing multiple processors should have value here. When the numerical analysis technique uses data structures such as vectors and matrices, a computer providing for direct operation on such structures will be of interest. However, there are key applications that simply do not lend themselves to either parallel or vector formulation. For such studies, a computer should provide very fast scalar floating-point arithmetic. Finally, for those studies where both parallel and vector formulations are possible, a computer providing all capabilities should be best.

Figure 1 Problem space: The domains of scalar, vector, and parallel implementations



History of large-scale engineering/scientific processors

The first computers did not have floating-point arithmetic in their primitive instruction set. The IBM 704 computer introduced this feature *circa* 1954,[2] in response to the need established by the use of the predecessor 701 computer for scientific applications. Built-in floating-point instructions have been a fundamental requirement since then, and the major focus has shifted to the speed of floating-point arithmetic.

Scalar floating-point arithmetic. The speed of floating-point arithmetic can be improved, up to a limit, by the addition of hardware circuits. There is a theoretical minimum number of logic levels required for each primitive floating-point operation,[3,4] but there is a practical limit to the number of circuits that can be employed. Computers in the 1950s and 1960s were often compared on the speed of a floating-point multiply operation.

In the late 1950s, several proposals were put forward to improve the execution speed of floating-point arithmetic, using "pipelining." IBM's 7030 (STRETCH) system,[5] for example, segmented a floating-point operation into sequential steps. Each step required one machine cycle; thus n machine cycles were required to complete an operation consisting of n steps. The STRETCH machine permitted the initiation of a new operation each machine cycle, so that step n of an operation would occur on the same machine cycle as step $n-1$ of the succeeding operation. The result was a possible execution rate of one operation per machine cycle. Achieving that rate was dependent upon a sequence of consecutive floating-point operations thought to be characteristic of engineering and scientific applications.

Vector floating-point arithmetic. The usefulness of pipelining on STRETCH (and other machines) depended upon a consecutive sequence of floating-point operations. Given that such sequences *could* occur, a machine designed with an architecture which guarantees that such sequences *do* occur should have a faster floating-point execution. This thinking led to the first vector machines, e.g., the CDC STAR-100[6] and the Texas Instruments ASC,[6] early in the 1970s. Vector instructions were made a part of the architecture, such that a guaranteed number of floating-point operations in a guaranteed sequence were presented for execution. The vector machines could execute a sequence of floating-point operations faster than any scalar machine of the day, provided the application used the vector instructions.

Parallel floating-point arithmetic. Another way to achieve a fast floating-point execution rate is to provide parallel computation elements. The ILLIAC IV,[7] also a machine of the early 1970s, provided for multiple floating-point operations to be completed simultaneously under the control of a single instruction. The ILLIAC machine could execute floating-point operations faster than any scalar machine of its day, provided the application used the parallel elements.

Current requirements for a large-scale engineering/scientific processor

Fast floating-point. The typical engineering/scientific application executes floating-point operations on 64-bit floating-point words, and the time to complete the application is dominated by such operations. It is therefore a requirement to make the floating-point execution speed as fast as possible. A NASA-sponsored survey[8] of the research community on the needs of scientific applications in the 1985–1990 time frame produced a requirement, based on a consensus of the respondents, of from 100 to 1000 million floating-point operations per second (MFLOPS). The requirement is not qualified with regard to whether this MFLOPS performance is needed at the job level or at the subroutine loop level. It is instructive to consider the implications of this requirement, with respect to the subroutine loop level, for the three different design approaches—scalar, parallel, and vector. As is argued below, no one approach to a large-scale processor (as opposed to a supercomputer) suffices for the subroutine loop level, and since this is an easier requirement to meet than the job level requirement, no one design approach currently meets the research community requirement.

For the purpose of understanding the three design approaches against the NASA-identified requirements, we assume a subroutine loop for a matrix multiply operation. The average number of cycles needed to execute floating-point operations in this loop can be derived for scalar and vector designs on the basis of previously published data.[9] With use of these derived data for a scalar design approach, a machine cycle of less than two nanoseconds would be needed to achieve 100 MFLOPS.[10] To meet the requirement on subroutine loops amenable to parallel processing, in a machine design of four parallel elements using four scalar elements of the assumed design, a cycle time of about seven nanoseconds would be needed to approach 100 MFLOPS.[10] To meet the requirement on subroutine loops amenable to vector processing in a single-pipelined vector machine of the assumed design, a cycle time of about twelve nanoseconds would be needed for 100 MFLOPS.[10]

Hence, each machine design, whether scalar, parallel, or vector, requires its own machine cycle to meet the requirement, scalar being the most demanding. The attainable machine cycle is in turn determined by circuit speed and packaging technology. Even the least demanding cycle time requirement, that of a single-pipelined vector machine, cannot be supported by the circuit speeds and packaging technologies used in the mid-1980s for large-scale processors (as opposed to supercomputers). The logical conclusion of this reasoning is that to approach the minimum requirements of the research community with a large-scale computer (as opposed to a supercomputer) in the mid-1980s, a parallel design in combination with a vector design is necessary.

A parallel design in combination with a vector design is necessary.

Other operations. The floating-point operation sequences that characterize engineering/scientific applications dominate the execution time, but other operations must also be performed. For one example, modeling boundary conditions requires special handling and uses logical and branching operations. For another example, data access methods are required to support the input and output of application data. The execution time of these "other" operations cannot be ignored in the design of a large-scale processor. If, for example, 75 percent of the time is spent in floating-point operations, the total time can be reduced at most by a factor of four if the 25 percent spent for "other" operations is not changed.

It is a requirement of large-scale engineering/scientific processors that these "other" operations be fast, lest the time to complete the nonfloating-point portion of an application erode the gains made in the floating-point portion.

Large memory. The NASA survey[8] that identified the need for 100 to 1000 MFLOPS in a 1985 – 1990 product also identified the need for 1 to 100 million 64-bit words of "core" memory and up to 1000 billion words of on-line storage. As a practical matter, the memory must be affordable, and, of course, must be fast enough to support the fast floating-point execution speed.

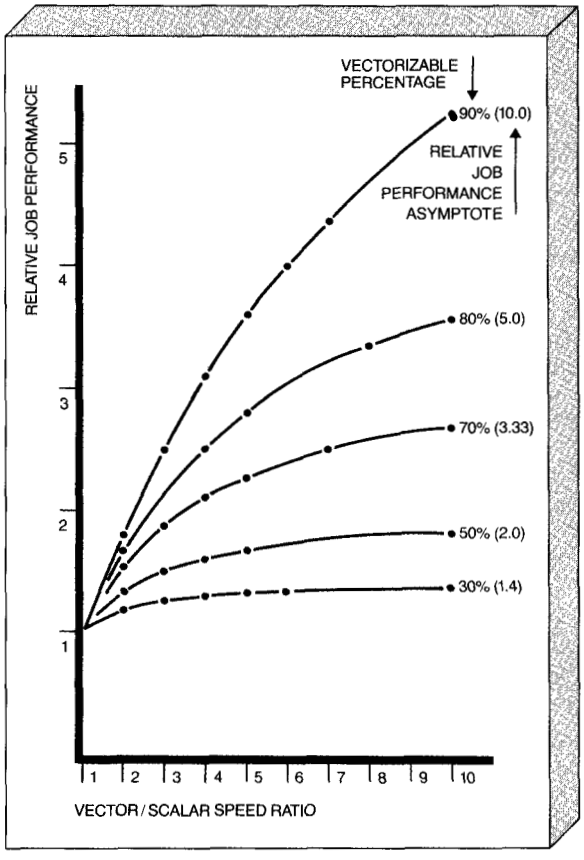
The requirement for "core" memory to have the attributes of being large (1-plus million 64-bit words), fast, and affordable suggests a hierarchy of memory elements. Many survey respondents specifically suggested a memory hierarchy of two or more levels. This is because fast memory is known to be inherently expensive, whereas large affordable memory is typically slow. A combination of some amount of fast memory with a larger amount of affordable memory, in a hierarchy, is a practical response to the requirement. It is reasonable to infer from the respondents' comments that system management of a hierarchy as directly addressable virtual storage is desirable.

Engineering/scientific design objectives of the 3090

The IBM 3090 processor is, like its predecessors (the 308X and 3033), a large-scale data processing system. These systems have been used by engineers and scientists for the management of their technical data, for the creation and display of graphics, and for interactive end-user computing. An additional design objective for the 3090, established by consideration of the current requirements for a large-scale engineering/scientific processor, was to integrate new performance capabilities into the base processor.

A design objective of this large-scale computer was to have reliability characteristics equal to or better than those of its predecessors and to be manufacturable in large quantities. This objective in turn dictated the use of a circuit technology package known to have high reliability and manufacturability characteristics. For IBM in the mid-1980s, the chosen emitter-coupled-logic circuit logic and thermal-conduction-module packaging determined that a machine cycle time of 18.5 nanoseconds would result. This scalar cycle time, in turn, dictated the need for both parallel and vector capabilities if the engineering/scientific requirements were to be approached. Since the scalar

Figure 2 Vector content effects



and parallel capabilities are also useful for general-purpose computing, these capabilities were to be included in the base offering. The vector capability, useful primarily for large-scale engineering/scientific applications, was to be an optional offering.

The objectives to provide scalar, parallel, and vector capabilities in the 3090 required new architecture as well as the use of pipeline and parallel implementation techniques. The existing IBM System/370 and 370/XA (Extended Architecture) architectures were scalar, with provision for parallel processing implementation. Only by extending these architectures[11] would it be possible to implement vector capability. However, an architectural extension alone would not be sufficient to create vector capability with the desired performance improvement. To achieve this capability, a pipeline implementation technique would have to be used. Parallel capability for a single engineering/scientific application would be provided by adding software to the already known parallel hardware implementation techniques.

Scalar objectives. From an engineering/scientific viewpoint, the scalar design of the 3090 had the objective to provide fast floating-point arithmetic. Matrix operations were selected as a specific measure of the ability of various proposed designs to meet the objective. The well-known matrix multiply operation and the matrix factorization operation were established as test cases for the scalar design. In the FORTRAN language, these test cases consist of DO-loops containing two operators, a multiply and an add. Thus, the design of the scalar 3090 focuses on high-speed execution of such loops.

Parallel objectives. Multiprocessor configurations have been a part of the IBM large-system product line for many years. Their purpose has been primarily to improve the throughput under a single operating system. In the engineering/scientific area, a parallel design would provide a theoretical capability to improve the turnaround time of a single application. The 3090 design objective was to provide both parallel hardware and supporting software such that an engineering/scientific application could be run in the minimum possible elapsed time. Fluid dynamics techniques were selected as a specific measure of the ability of the design to meet the objective. Since it was understood that the scalar engine would be replicated in a tightly coupled configuration to form the parallel offering, attention was concentrated on software to exploit the capabilities of the hardware. The objective was established to have the MVS multitasking capability surface at the FORTRAN level in a user-friendly way.

Vector objectives. Given the IBM 3090 high-speed base processor, the next step was to consider how to add a vector processing capability to further enhance performance on appropriate applications. A wide variety of performance choices were available, and, as expected, cost increased with performance. The choice of the appropriate performance target, and the associated cost, was a function of the nature of the applications that were considered of primary importance.

A study of many engineering and scientific applications revealed that certain portions of the existing scalar programs could be profitably replaced

by vector functions. The time required to execute these replaceable portions, divided by the total scalar run time, is called the vectorizable fraction, which can also be expressed as a percentage. The time required to execute the vectorizable fraction on a scalar element, divided by the time to execute the same function on a vector element, is called the vector/scalar speed ratio. The machine designer can choose to speed up the vectorizable fraction by a large or a small amount. The decision should be dictated, at least in part, by the applications for which the product is intended.

As Figure 2 shows, the relative job performance that can result from adding vector capability depends both on the vector/scalar speed ratio and on the vectorizable percentage of an application. Each curve in the figure can be derived using the equation

$$\text{Relative job performance} = 1 / [(1 - F) + F / VSR]$$

where F equals the vectorizable fraction and VSR equals the vector/scalar speed ratio.

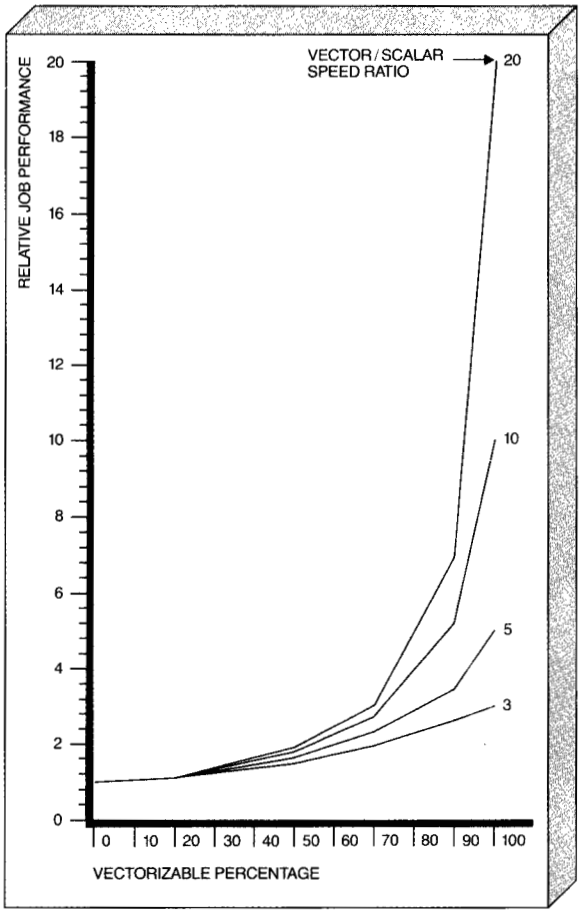
Applications with a 90 percent vector content could approach a ten-times speedup as the vector/scalar speed ratio reaches a high level. However, applications with only a 30 percent vector content would achieve most of their gain from a rather modest vector/scalar speed ratio.

The choice of the appropriate vector/scalar speed ratio should be determined by the vector content of the anticipated set of applications.

Figure 3 is another plot of the same equation, this time using F as the independent variable. The individual curves representing different vector/scalar ratios become widely divergent at high vectorizable percentages. However, in the midrange of vectorization the spread in performance is not as pronounced.

Extensive studies of the projected set of applications for the IBM 3090 Vector Facility led to the decision to optimize the design for the midrange of vectorizable percentage. Interpretation of Figures 2 and 3 led to a vector/scalar speed ratio goal in the vicinity of four. This ratio would result in a very cost-efficient design providing job performance gains quite comparable to those of other designs with much higher vector/scalar speed ratios at significantly higher cost.

Figure 3 Vector/scalar performance effects



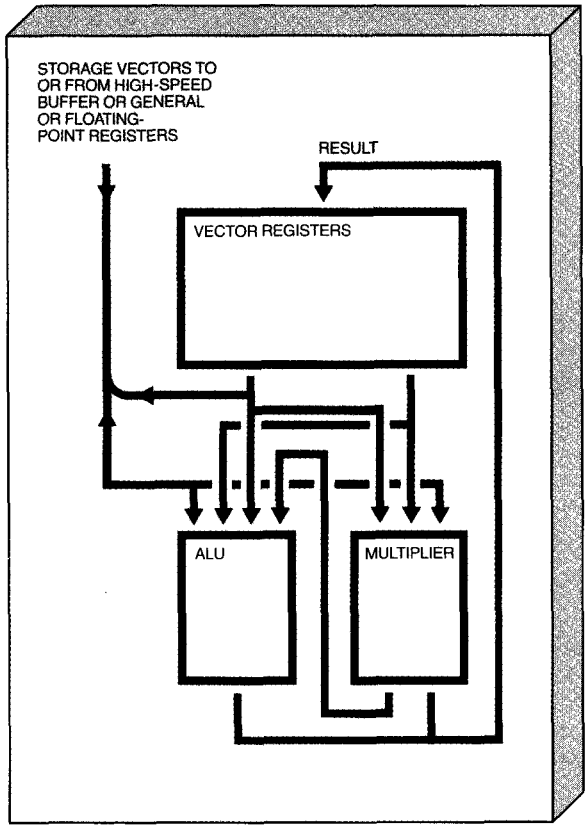
The 3090 offering that evolved from the above scalar, parallel, and vector design objectives is described in the remainder of this paper.

Engineering/scientific features of the 3090

The 3090 offers an approach to the current requirements for a large-scale engineering/scientific processor by providing in one system a high-performance scalar capability, dyadic and four-way parallel capability, and an optional vector capability.

Scalar features. Features of the scalar design include a high-speed multiply function, an improved

Figure 4 Vector Facility block diagram



add function, the elimination of address generation interlock in the loop-closing instructions, and a mechanism for conditional branch handling. These features are described in more detail in the paper by Tucker.[12] At the time of the 3090 announcement in February 1985, its scalar capability was the best listed[13] for scalar machines running MacNeal-Schwendler's NASTRAN.®

Parallel features. The February 1985 announcement included both a dyadic and a four-way offering, usable under FORTRAN for parallel processing. An innovative use of FORTRAN library routines in combination with MVS multitasking provided the support needed for the use of all available resources on a single engineering/scientific application. Benchmark tests showed the capability to reduce elapsed time by up to 1.8 times on the dyadic and up to 3.3 times on the four-way when compared to a single processor. The software support for this is described in more detail later in this paper.

Vector features. The October 1985 announcement of the Vector Facility for the 3090, one per processor, offered a machine capable of approaching the minimum requirement of the research community on routines suitable for parallel vector processing. On routines (e.g., matrix multiplication), the best achievement on a single Vector Facility was about three quarters of the minimum requirement of the research community. When an application is able to use independent routines on the parallel vector hardware, it is possible to surpass the minimum requirement.

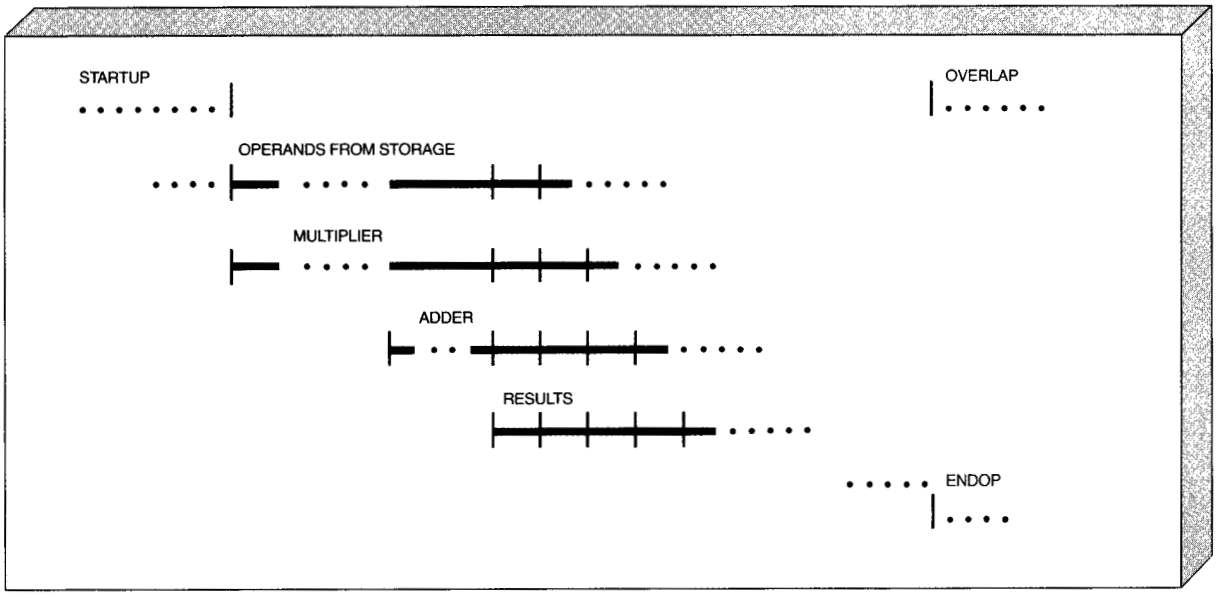
The Vector Facility achieves the cost-performance design objectives by use of a powerful instruction set, a pipelined multiplier and ALU, and a design integrated with the base IBM 3090, including its high-speed cache to provide operand data. These features are described in more detail in a succeeding portion of this paper.

Memory. The 370/XA architecture permits direct addressability of a virtual memory of 256 million 64-bit words, consistent with the large memory requirements for a current large-scale engineering/scientific processor. The IBM 3090 implements physical storage to realize this virtual addressing architecture with a hierarchy of high-speed buffer, central storage, expanded storage, and DASD. This provides the flexibility to use both a high-performance chip technology in the central storage and an advanced, low-cost, dense chip technology in the expanded storage.

Vector Facility highlights of the 3090

A vector processing capability has been integrated into the basic structure of the 3090,[12] fundamentally as an extension of the execution element in each of the central processors (CPs). The instruction element of the central processor has the capability to decode the vector instruction set, consisting of 171 instructions, and direct the execution, depending on the instruction type, to either the execution element or the vector element. The buffer control element of the central processor also participates in vector instruction handling, just as it does with nonvector instructions.

Figure 5 Vector multiply and add sequence chart



A Vector Facility can optionally be added to one or more of the central processors in the IBM 3090 models.

Data flow. The IBM 3090 Vector Facility data flow is shown in Figure 4.

The 8 (or 16) vector registers, each holding 128 elements, can supply up to two 64-bit (or 32-bit) word operands per machine cycle. These operands can be delivered to either the Arithmetic Logic Unit (ALU) or the Multiplier, or one operand to each. The ALU and the Multiplier can also be supplied via a data path that carries operands from storage by way of the high-speed buffer, or from the general or floating-point registers. Another data path into the ALU comes directly from the Multiplier. This path supports the compound vector instructions, such as multiply and add, generating as many as two floating-point operations per machine cycle.

Pipeline operation. The timing of the pipelined execution of a vector instruction is variable. Each vector instruction may be thought of as composed of four sequential parts. These are startup, vector execution, end-of-operation (end-op), and overlap. Each part requires a specific number of machine cycles, depending on the specific vector instruction.

The complexity of four sequential parts to each instruction, and the variability of timing for each part, introduce the need for a first-order approximation to the timing of vector instructions. For this purpose, a matrix multiply code has been examined, each sequential part has been analyzed, and the whole has been reduced by combining startup, a portion of the vector execution, the end-op, and the overlap. This combined timing has been prorated across the vector instructions that constitute the matrix multiplication code, and for this purpose has been called simply vector overhead. The resulting first-order approximation of the time to execute a vector instruction is 28 cycles of vector overhead plus one cycle per element, per vector instruction.

The sequence of a smoothly flowing pipeline during vector execution, doing a multiply and add vector instruction which performs two floating-point operations to produce one result, is shown in Figure 5. Note that one result is produced per machine cycle, with the registers and storage supporting the flow of three operands per cycle.

The Multiplier unit, which consists of three separate Multipliers, each capable of producing a product three cycles after receiving the input, can accept a new multiplier/multiplicand pair each cycle.

The two streams of operands, one from storage and the other from a floating-point register, are initiated. Several cycles later the first product emerges from the Multiplier and is fed to the ALU for the ADD. On that same cycle the other input to the ALU is supplied with the operand from a vector register which represents the running sum. A few cycles later the first of the results is produced, ready to be gated into a vector register. The operation continues, one result per machine cycle, until the vector length is exhausted (or the section size, whichever occurs first).

The depicted multiply and add sequence is illustrative of operations within the Vector Facility.

Section size. The length of a vector may be shorter or longer than the 128 elements available in a 3090 vector register, as determined by the application. The 370/XA vector architecture provides special instructions to simplify the processing of variable-length vectors on fixed-length registers. The application specifies the length of the vector to be processed, and the processor divides the processing of the vector into "section-size" pieces (128 elements in the case of the 3090). If the vector is longer than the section size, the processor executes a "section" at a time. If shorter, the processor stops when the last element has been processed. The section size is chosen by the processor designer.

Large section size increases cost and save/restore time, but reduces startup effects. The chosen vector section size of 128 was based upon cost, performance, and application considerations. The applications considered did not benefit significantly from a section size greater than 128 elements.

Memory hierarchy. An important implementation consideration for the design of a vector facility is the provision for accessing high-speed storage. Two approaches may be considered: (1) a single-level, very high-speed, high-bandwidth central storage, or (2) a memory hierarchy providing a very high-speed cache.

In the 3090 the memory hierarchy approach was selected for several reasons. First, there is a definite cost advantage to implementing a high-speed cache, backed by a large central storage, as compared to a single, large, high-performance central storage. The cache concept, introduced in 1968,[14] is generally used in the implementation of large-scale computers. With the introduction of expanded storage in the 3090, the cost advantage accrues a second time. Expanded storage provides capacity extensions with performance approximately equivalent to that which would be attained by the same capacity extension of central storage, but with reduced cost.

Second, because a high-performance scalar processor is required in conjunction with the Vector Facility, it is appropriate to share the mechanisms, and thereby share the costs, of the memory hierarchy. And third, there is a favorable performance characteristic when using the Vector Facility with a memory hierarchy. More specifics on the performance of the Vector Facility are contained in Part II of the paper by Clark and Wilson.[15]

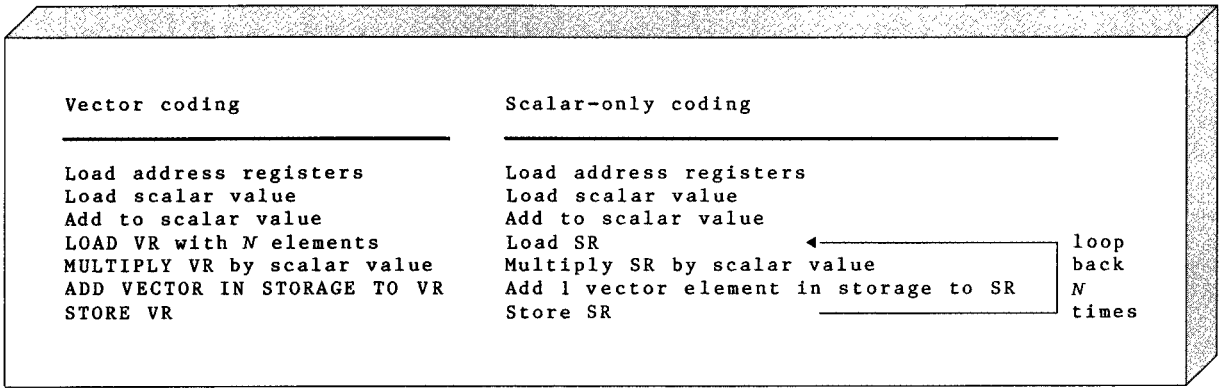
Engineering/scientific software support for vector processing

Requirements. An application can benefit from the speed of the Vector Facility only if it contains vector instructions. A sample of conceptual object level code of an application using vector instructions might look like the code depicted on the left in Figure 6. The corresponding sample of object code using only scalar instructions would look like the code depicted on the right.

The example shows vector instructions in capital letters (for ease of reading the example) and uses the abbreviations VR for Vector Register and SR for Scalar Register. On the left side the example contains both vector and scalar instructions, typical of vector coding. The same scalar instructions appear on the right in the scalar-only coding, but here the four vector instructions have been replaced by four scalar instructions and an indicated loop. To obtain the same result with both codings, the 3090 would have to loop through the scalar-only coding as many times as there are elements in the vector being processed, whereas only one pass is required through the vector coding.

If no vector instructions are present in the executed object code, the Vector Facility will remain idle. Two methods are available to utilize the Vector Facility. One method is to use prepackaged programs that already contain vector instructions. Another method is to incorporate vector code into a program using a vectorizing compiler, an assembler supporting the vector instructions, or a "call"

Figure 6 Hypothetical object code



to a subroutine which utilizes the vector instructions.

Available software support. A basic area of software support is that of prepackaged programs from independent software vendors. Studies conducted by IBM have shown that in many large-system installations doing engineering and scientific applications, over 50 percent of the production workload consists of the running of these prepackaged programs. A number of these programs already exist for IBM scalar processors, and IBM is encouraging and cooperating in the conversion of as many of them as is feasible to vector and parallel versions.

Also fundamental to the support of the Vector Facility is a VS FORTRAN offering which has the capability to take existing programs written in IBM FORTRAN, both ANSI level 66 and 77, and to compile them, producing vector object code where such potential exists. FORTRAN DO-loops will be converted to vector instructions where possible and reasonable. Other conditions, for example, IF statements that cannot be removed or accommodated, may inhibit vectorization.

A critical part of every FORTRAN facility is an efficient FORTRAN library. Many of the subroutines in the FORTRAN library now have vectorized versions and enhance the performance of the system. The Interactive Debug Facilities (IAD) are also provided as part of the library. (This same library provides the Multitasking Facility.)

Extensions to the Assembler H product are provided to support the 171 new vector instructions.

An Engineering and Scientific Subroutine Library (ESSL) is also available. Subroutines covering a variety of methods are included: basic linear algebra subprograms, routines for the solution of simultaneous linear algebraic equations for a variety of conditions, eigenanalysis routines for a real general matrix and a real symmetric matrix, signal processing routines including fast Fourier transforms and IBM 3838 array processor algorithms, matrix operations, a random number generator, and an error monitor.

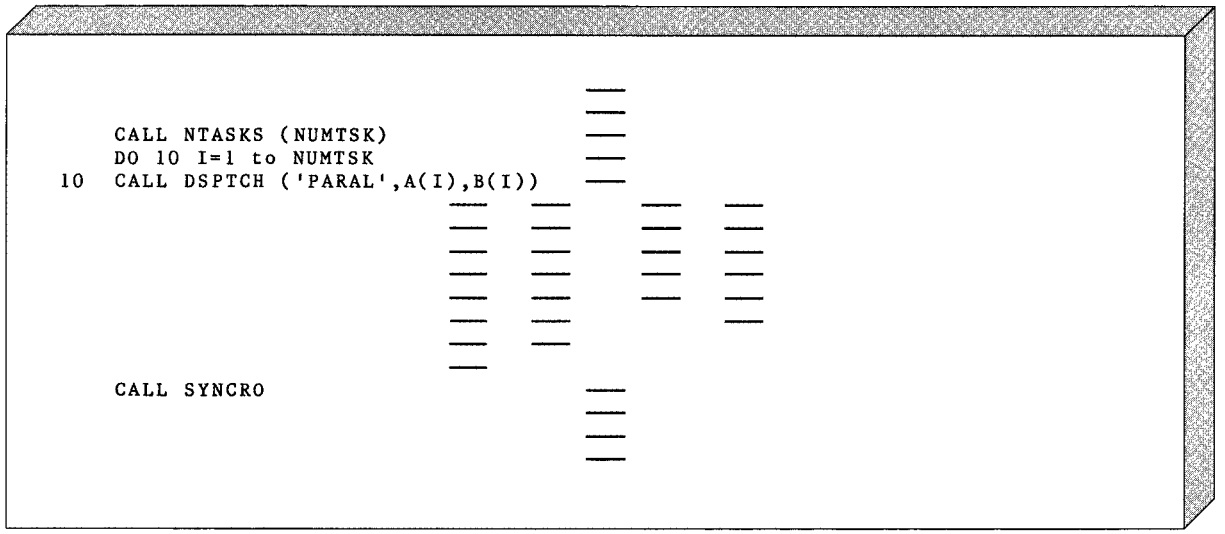
For the seismic analysis application, the Vector Processing Subsystem/Vector Facility (VPSS/VF) allows users of the IBM 3838 Array Processor to utilize the Vector Facility as an emulator of their current VPSS programs, without rewrite, with an improvement in performance. A scalar option within VPSS/VF can also be used for emulation, with lower performance.

For applications that experience heavy usage or are of vital importance to an installation, it may be both desirable and profitable to re-examine the application to determine if it might be restructured to use the above vector software support with the VS FORTRAN Program Multitasking Facility.

Engineering/scientific software support for parallel processing

As stated in the design objectives, the focus for parallel capability was on software. A set of three

Figure 7 Multitasking flow



subroutines was developed to further improve turnaround performance for those users having appropriate critical applications. As a part of the VS FORTRAN library, this set of subroutines, known as the VS FORTRAN Program Multitasking Facility, was provided to enable a user with a dyadic or four-way system to structure a program so that those parts of an application which can be run in parallel (inherent parallelism) on the multiple processors may do so. These subroutines allow the application programmer to utilize the multitasking capability of MVS and to thus reduce the "turnaround" time of the application. They allow the concurrent scheduling ("fork") of those portions of the application which permit it, and also provide the ability to synchronize ("join") the completion of the parallel parts when it is necessary to ensure the completion of all prior work before a program can continue. Figure 7, depicting time from top to bottom, with statements executed as bars, is an example.

In the example, the three subroutines that constitute the application interface are shown. The first subroutine, CALL NTASKS, returns to the application the number of subtasks available for concurrent scheduling. In the example, the value 4 is returned in the argument NUMTSK, indicative of four available subtasks (as might be preferred on a 3090/400). The second subroutine, CALL DSPTCH, attaches one application subroutine for execution by one subtask on one available processor. In the example, the application subroutine PARAL is dispatched four times, each time with an independent set of arguments A(I) and B(I), for execution on the four available processors. Although not shown by this example, it is also possible to dispatch distinct application subroutines. The third subroutine, CALL SYNCRO, causes the main program to wait until all concurrently scheduled tasks have completed. In the example, this is pictorially shown by the middle column of bars representing the main program and by the side columns representing the four dispatches of the application subroutine PARAL.

Engineering/scientific performance of the 3090

A broad range of applications across a wide spectrum of industry has developed, and it is at these areas that the IBM 3090 Vector Facility is directed. The application areas listed in Figure 8 are examples of those that should benefit from the combined capabilities of the IBM 3090 system with its excellent scalar performance enhanced, where appropriate, by both the Vector Facility and the VS FORTRAN Program Multitasking Facility.

The current approach to many of the application programs in Figure 8 is still based on conventional scalar processing. The future should see growth both in the number and type of applications of interest, and a trend toward vectorization and parallelization of these applications. This should

occur as the benefits of these techniques, in both improved performance and lowered computer costs, become apparent.

An example. The October 1985 announcement of the 3090 Vector Facility identified certain performance data for a set of applications. As an example, the levels of performance attained for an engine design application, turbine blade analysis (T-Blade), are expanded in Figure 9. This figure includes internal throughput rate (ITR) ratios and external throughput rate (ETR) ratios,[16] using the 3081KX with one central processor (CP) as the reference point. Further discussions of performance are contained in Part I of the paper by Clark and Wilson.[15] The steps required to attain the performance values given in Figure 9 are illustrative of scientific and engineering processing on the 3090/200 with Vector Facility.

The T-Blade application is written in FORTRAN. The base 3081KX and the 3090/200 one-CP scalar values were determined by compiling the application source code under VS FORTRAN Version 2, producing scalar code using the highest level of scalar optimization. The resulting source code was then executed and timed[17] and its internal throughput rate (ITR) ratio calculated.

Next the same application source code was compiled using the highest level of vector optimization, which causes the compiler to automatically produce object code with vector instructions where possible. The resulting source code was then executed and timed[17] on a single CP and Vector Facility, and the performance calculated. This calculation produced an ITR ratio of 5.2, which, while respectable, was judged to be a performance level that could be improved upon. The original source code was examined and the inner loops, as written in FORTRAN, were modified according to "good vector coding practices." Some of these coding practices are described in a comprehensive report by Dubrulle et al.[18] The new source code was then compiled, executed, and timed[19] to produce the 6.9 ITR ratio given in Figure 9.

The final step involved another modification to the FORTRAN source code, this time to create multiple subroutines that could be invoked using the Multitasking Facility. This new source code was then compiled to produce scalar code, and again to produce vector code. In both compilations the highest level of optimization was used. The compiled codes were then executed on the 3090/200. Since parallel processing performance is a measure of turnaround time, which is based on elapsed time, the elapsed times were recorded. Therefore, for the parallel processor values, external throughput rate ratios were calculated.

The T-Blade application is one of a class of applications that can benefit from all of the capabilities (scalar, parallel, and vector) of the 3090. All floating-point applications will, of course, benefit

Figure 8 Examples of suitable applications

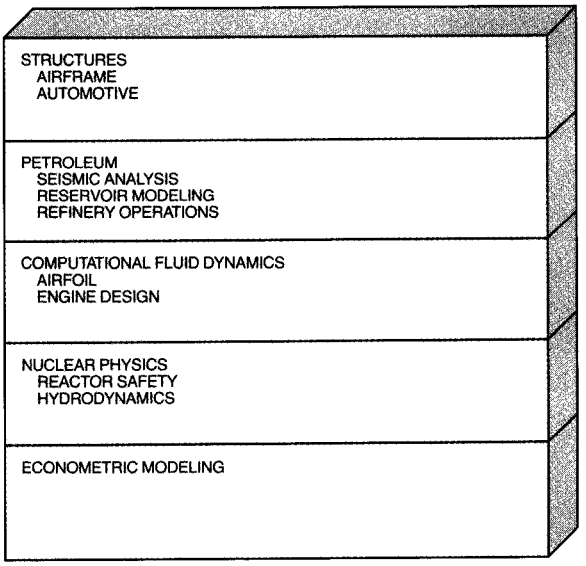
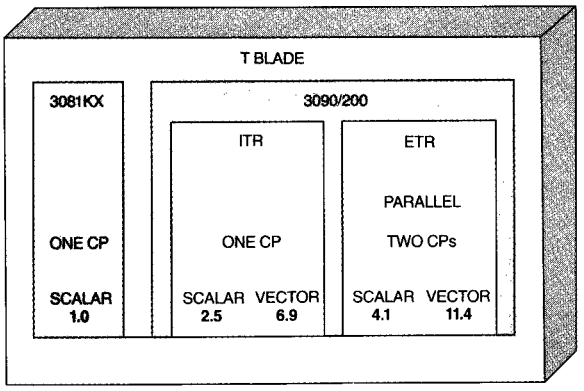


Figure 9 3090 processor, Model 200: T-blade performance in ITR and ETR ratios



from the scalar capability. Some applications may experience improved performance from the use of vector compilation and execution as described above. Some applications may experience improved performance from source-code modifications to invoke the parallel capability. Further, some applications may experience improved performance from using the ESSL library, a possibility previously discussed but not illustrated by the T-Blade application.

Matrix multiplication routines are particularly interesting.

Engineering/scientific performance parameters. The performance of the IBM 3090 with Vector Facility varies as a function of specific application characteristics and coding techniques. A few of the most important performance parameters follow. An understanding of these parameters may be useful to the FORTRAN language user, the assembler language user, or both.

Contiguous data: Although scalar performance will vary with the placement of data in memory, this parameter is particularly important in vector performance. The rule is to access contiguous (e.g., column-order for FORTRAN) application data where possible.

Reuse of data: Data present in cache or registers (general-purpose, floating-point, or vector) should be reused where possible.

Vector length: The rule is to provide long vectors rather than short vectors.

Concurrency: The rule is to use those algorithms and coding techniques that will give the maximum concurrent usage of the available hardware facilities. The hardware facilities that may be used concurrently are the dyadic or four-way CPs, including their vector facilities. Performance increases as the amount of concurrent usage, sometimes referred to as the inherent parallelism and the vectorizable percentage, increases.

Instruction mix: Generally speaking, in the execution of scalar floating-point operations, addition is faster than multiplication, and both are significantly faster than division. In the execution of vector floating-point operations, compound operations, such as multiply and add, are faster than the corresponding separate vector instructions, and all are significantly faster than division. The rule is to remove division from the inner loop wherever possible, and when dividing more than once by the same number, to obtain an inverse and multiply.

Knowledge of these 3090 performance parameters has proven useful in construction of the code for routines in the Engineering and Scientific Subroutine Library. The matrix multiplication routines, for example, are particularly interesting because of their level of attainment. Consideration was given in the design of these routines to the section size of the 3090 Vector Facility, the available compound instructions, and the cache size. The levels of performance attainment for a subroutine multiplying matrices with order between 100 and 1000 may be derived from previously published material.[20] The LOOP_MFLOPS values so derived are applicable only to this subroutine loop. For matrix multiplication on one 3090/200 Vector Facility, the derived value is around 70 LOOP_MFLOPS, depending on the size of the matrices. When two or four Vector Facilities are executing copies of the matrix multiplication routines, the capacity of the system increases approximately with the number of Vector Facilities. Of course, these values apply to routines, not to full jobs. JOB_MFLOPS values are considerably less than LOOP_MFLOPS values. The reader should refer to the paper by Clark and Wilson[15] for specific examples of JOB_MFLOPS on the 3090/200 Vector Facility.

Summary

Engineering/scientific applications may be addressed in a large-scale processor by three complementary approaches:

1. High-speed scalar floating-point processing
2. Parallel processing
3. Vector processing

The IBM 3090 processor offers a balanced response to these three approaches by including a high-performance scalar capability; dyadic and four-way multiprocessing, which in conjunction with the VS FORTRAN Program Multitasking Facility provides a parallel processing capability; and an optional Vector Facility which provides a vector processing capability.

The design of the Vector Facility was based on application studies which concluded that application vectorizable percentages significantly beyond the midrange, though possible, are not currently anticipated to be the norm. This implies that a Vector Facility should be accompanied by a high-performance scalar capability.

The base IBM 3090 achieves its engineering/scientific performance enhancements through improvements in specific floating-point and branch instruction handling. The 256 million 64-bit-word virtual storage is implemented in a physical storage consisting of a high-speed buffer, central storage, optional expanded storage, and DASD.

The Vector Facility achieves its cost-performance design objectives by use of a powerful instruction set, a pipelined multiplier and ALU, and a design integrated with the base IBM 3090, including its memory hierarchy and high-speed buffer.

Extensive software support has been provided with the Vector Facility, including operating system support, a vector FORTRAN compiler, a FORTRAN library, and an engineering/scientific subroutine library.

Acknowledgments

The comprehensive acknowledgment of the individuals on the teams responsible for the IBM 3090 would be impractical, for they are numerous. We wish, rather, to recognize the teams by function. The architectural team coordinated the many aspects of definition to develop the integrated vector architecture upon which the IBM 3090 Vector Facility is based. The planning team identified the applications, the section size requirements, and the performance and price-performance goals. The design team established the design direction leading to the specific implementation of the IBM 3090. The development team integrated the logical and physical elements necessary to realize the final packaged product and conducted the verification testing. The programming team planned, designed, and developed the supporting software. We wish also to acknowledge the leadership of the management team in bringing the IBM 3090 to the marketplace.

NASTRAN is a registered trademark of the National Aeronautics and Space Administration. MSC/NASTRAN® is an enhanced proprietary version developed by the MacNeal-Schwendler Corporation.

Cited references and notes

1. Scalar processing operates on scalars, which are single data items. Parallel processing operates on either scalars or vectors, utilizing multiple processors on a single application. Vector processing operates on vectors. A vector is a collection of data items which is ordered along a single dimension.
2. R. Moreau, *The Computer Comes of Age*, MIT Press, Cambridge, MA (1984).
3. S. Winograd, "On the time to perform addition," *Journal of the ACM* **12**, 277 (1965).
4. S. Winograd, "On the time to perform multiplication," *Journal of the ACM* **14**, 793 (1967).
5. W. Buchholz, *Planning a Computer System—Project Stretch*, McGraw-Hill Book Co., Inc., New York (1962).
6. P. Kogge, *The Architecture of Pipelined Computers*, McGraw-Hill Book Co., Inc., New York (1981).
7. K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill Book Co., Inc., New York (1984).
8. The survey was distributed to about 1800 individuals, of whom about 10 percent responded, representing 90 affiliations. The survey and results are summarized in a paper by Peter Lykos, "Working Document on Future Computer System Needs for Large Scale Computations," Joint Project of NASA Ames Research Center and IIT, 1978.
9. Contained in Table 3 on page 14 of the October 1985 IBM 3090 Product Announcement Letter 185-120; available through IBM branch offices.
10. Cycle time in microseconds = N/(MFLOPS × cycles per floating-point operation), where N = number of independent processing elements.
11. W. Buchholz, "The IBM System/370 vector architecture," *IBM Systems Journal* **25**, No. 1, 51–62 (1986, this issue).
12. S. G. Tucker, "The IBM 3090 system: An overview," *IBM Systems Journal* **25**, No. 1, 4–19 (1986, this issue).
13. A number of engineering/scientific application packages available from non-IBM software vendors run on processors that have System/370 or 370-XA architecture, including the 3090. Some include a run time estimate for each processor on which the application will execute. MacNeal-Schwendler Corporation's MSC/NASTRAN,® one such package commonly used for structural analysis, provides an application manual. This manual tabulates a processor-dependent variable which can be used to estimate total job run time. This variable, called the M value, is the time in microseconds required for one execution of a floating-point multiply/add loop on a given processor. The M values are sometimes

used as a rating of a processor, and comparison of M values can be made among processors. In such a comparison, the smaller the M value, the faster the processor. A MacNeal-Schwendler Corporation document entitled *Time Estimation and Problem Execution* contains information published in the applicable computer-dependent editions of the MSC/NASTRAN Application Manual (Section 7.3). Included in the document are the M values for processors from the many vendors for which the MSC/NASTRAN software package is available. The February 1985 edition lists the 3090 Processor Unit Model 200 M value as smaller than that of any other nonvector processor cited. That is, the 3090 Model 200 executes the MSC/NASTRAN multiply-add loop faster than any other nonvector processor listed in the publication.

14. C. J. Conti, D. H. Gibson, and S. H. Pitkowsky, "Structural aspects of the System/360 Model 85; Part I, General organization," *IBM Systems Journal* **7**, No. 1, 2 – 14 (1968).
15. R. S. Clark and T. L. Wilson, "Vector system performance of the IBM 3090," *IBM Systems Journal* **25**, No. 1, 63 – 82 (1986, this issue).
16. For those who may be unfamiliar with the terms ITR ratio and ETR ratio, refer to K. Radecki, *Introduction to Processor Performance Evaluation*, IBM Washington Systems Center Technical Bulletin, GG66-0232, IBM Corporation (February 1986); available through IBM branch offices.
17. Refer to Table 1 on page 11 of the October 1985 IBM 3090 Product Announcement Letter 185-120; available through IBM branch offices.
18. A. A. Dubrulle, R. G. Scarborough, and H. G. Kolsky, *How to Write Good Vectorizable FORTRAN*, G320-3478, IBM Corporation; available through IBM branch offices.
19. Refer to the IBM 3090 Vector Performance Bulletin, GG66-0245; available through IBM branch offices.
20. Refer to Table 3 on page 14 of the October 1985 IBM 3090 Product Announcement Letter 185-120; available through IBM branch offices. Note that $(2 \times N^3)$/(time in microseconds), where N is the order of each of the two square matrices being multiplied, is approximately equal to LOOP_MFLOPS.

Donald H. Gibson *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Mr. Gibson joined IBM in 1956, specializing in CPU and memory design, working first on the SAGE project and then on the STRETCH system. After attending IBM's Systems Research Institute in 1962, he did early design study work leading to the System/360 Model 91 and System/370 Model 195. His simulation work on block transfer memory systems design led to the "cache" design, first introduced in the System/360 Model 85. For his contributions, he shared a Corporate Outstanding Contribution Award in 1967. He was manager of the Large Systems Technical Support Group in the late 1960s and early 1970s, then left management in the mid-1970s to pursue technical interests in graphics and artificial intelligence. His customer survey consultancy in the late 1970s led to the engineering/scientific design features of the 3090, including the Vector Facility. In the early 1980s, he led the activity in IBM on engineering data base, chaired the work on a computer-aided engineering strategy, and pioneered the work on cooperative processing. Mr. Gibson is currently IBM Poughkeepsie's senior engineer for Engineering/Scientific Requirements. In this position he led the initial effort to define and announce the scalar engineering/scientific capabilities of the 3090. He earned the B.S. in electrical engineering at the University of Kentucky in 1956.

Don W. Rain *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Dr. Rain is a Senior Technical Staff Member in Advanced Processor Development. He joined IBM in 1964 at Poughkeepsie and has since been involved in multiple aspects of large-system design. He was involved with design support activities including performance analysis and simulation of large systems until 1971. From 1969 to 1975 he was involved with a high-level machine architecture definition, user specialty language creation, and an application development system architecture. From 1975 to the present he has had responsibilities in the area of overall system design of IBM's high-performance processors, including specifically the IBM 3090. Areas of concentration have included system functional specification, storage hierarchy definition, and project coordination. In 1985 he received an IBM Outstanding Innovation Award for his work on expanded storage. Dr. Rain received a B.S. degree in electrical engineering from Purdue University in 1958, an M.S. degree from the University of Connecticut in 1961, and a Ph.D. degree in electrical engineering from the University of Illinois in 1964.

Hugh F. Walsh *IBM Data Systems Division, P.O. Box 100, Kingston, New York 12401.* Mr. Walsh (now retired) was most recently Program Manager of Engineering/Scientific Analysis for IBM's Kingston Laboratory. He has been with IBM since 1955 in technical marketing, education, and systems planning. Since 1960 he has been involved in the planning of IBM's large systems, including the System/360 Models 91, 95, and 195 and most recently the IBM 3090 and the Vector Facility. His experience has led to his involvement in computer systems in a wide variety of application areas including petroleum, aerospace, automotive, and circuit design usage. Mr. Walsh holds a B.S. in physics from Siena College and an M.S. in physics from Rensselaer Polytechnic Institute.

Reprint Order No. G321-5260.