Design and performance of the IBM Enterprise System/9000 Type 9121 Vector Facility

by T. J. Slegel
R. J. Veracca

The design of the IBM Enterprise System/9000™ Type 9121 Vector Facility is described and its performance is evaluated in this paper. The Vector Facility design adheres to the architecture developed for the 3090™ vector facilities. The original design objectives and associated architecture are reviewed. Vector operations and design details are discussed, and specific performance results are shown.

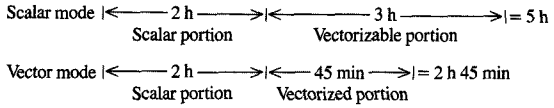
Introduction

The IBM Enterprise System/9000™ (ES/9000™) Type 9121 Vector Facility implements the IBM ESA/390™ Vector Architecture [1] on a single air-cooled thermal conduction module (TCM). This vector facility is used in each of the ES/9000 models. In the uniprocessor models (190, 210, 260, 320), the vector facility is physically mounted in the main frame with the rest of the central processor (CP) and memory system, whereas on the dyadic models (440, 480) it is mounted in an expansion frame attached to the side of the main frame.

Although the design is based on that of the IBM ES/3090™[1] Model J Vector Facility, most areas were significantly redesigned to meet the challenges imposed by new technologies. The design goals for the vector facility were established during the development phase of the first IBM 3090™ vector facility [2], announced as a product in October 1985, and the framework for the evaluation of these product objectives was defined early in the design stage. These original objectives have been the criteria used for the development of the follow-on IBM vector facilities. Each generation of vector facility has brought with it some additional form of performance enhancement [3].

Because these original objectives are, for the most part, still applicable to the ES/9000 Type 9121 Vector Facility, the design objectives for the vector facility are reviewed in the first part of the paper, followed by a brief review of the vector architecture. Next, some essential aspects of the vector facility design are discussed, and an overview is presented of how a typical vector instruction operates along with the vector interface with the cache. Then, the method used to evaluate the vector facility performance is

[1] The 3090 Models E, S, J, and JH processor families are designated as ES/3090 models.



The vectorized portion of the job represents a four times increase over the same portion of the job run scalar. The resultant vector/scalar speedup ratio is 1.8.

Figure 1

Application execution times.

described, including some details about the performance measurement techniques. Finally, the results of these performance measurements are discussed.

Review of the vector design objectives

The IBM ES/9000 Type 9121 Vector Facility is an optional performance-enhancement feature available for each of the ES/9000 Type 9121 processors. This feature offers the ability to process vector instructions defined by the IBM ESA/390 Vector Architecture. This is accomplished by adding vector registers, vector arithmetic hardware computational elements, and other hardware to the processor. The objective of the vector facility was not to provide the fastest possible vector hardware, but rather to provide an integrated, balanced system solution. The basic goal was to complement a fast scalar processor. Many of the numerically intensive computing (NIC) application programs for which a vector processor is utilized may have considerable scalar content. Even highly vectorizable programs sometimes have portions of code which run more efficiently in scalar mode. The goal was to complement the fast scalar performance of the central processor with a level of vector performance three to five times faster for applications in the 50–80% level of vectorization, the mid-range of application vectorizability. This vector/scalar loop speedup ratio[2] of 3 to 5 provides internal throughput rate (ITR)[3] improvements in the range of one-and-a-half to almost three times the scalar performance for typical vectorizable applications. For example, assume that a processor has a vector performance capability four times greater than its scalar performance capability. Given a job that is 60% vectorizable and uses five hours of CPU time in scalar mode, the comparison shown in **Figure 1** can be made.

[2] CPU time improvement for the vectorizable portion of the job.
[3] ITR is defined as the number of jobs completed per central processor busy second.

Besides complementing the scalar performance, another design objective was to reach this level of vector performance without limiting dyadic processing capabilities. Dyadic processors can provide increased computing capacity to meet the demand of a multiple-job workload by using the VS FORTRAN Multitasking Facility (MTF), or they can improve turnaround time for a single job through parallel processing via the Parallel FORTRAN (PF) product.

Review of the IBM ESA/390 Vector Architecture

The IBM ESA/390 Vector Architecture is a common architecture across all models in the System/390™ family that support a vector facility. Its main features include

- 16 vector registers.
- Vector mask register.
- Vector status register.
- Vector activity counter.
- 171 instructions.

The Vector Architecture provides 16 vector registers (VRs) with 256 elements of 32 bits each. (The architecture allows the number of elements to vary on different models, but the ES/9000 Type 9121 Vector Facility uses 256 elements per VR.) An even/odd pair of VRs may be combined to form a logical vector register which is 64 bits wide. If all VRs are used in this manner, only eight logical VRs are available, but their usage may be mixed so that some registers contain double-word data while others contain word-size data.

The vector mask register (VMR) is 256 bits in length, with each bit corresponding to an element in a VR. If the VMR is used during a vector-add instruction, for example, and a bit in the VMR is zero, the corresponding element in the vector registers is not modified. The VMR is primarily used for conditional execution in programming loops and for sparse-matrix manipulations.

The vector status register (VSR) is 64 bits wide and contains several fields. For most instructions, the vector count (VCT) contains the number of elements that are to be processed. The vector interruption index (VIX) holds the number of the element currently being processed and is used for resuming execution after an interruption. The vector mask mode bit controls whether the VMR will be used in executing a general-type vector instruction. In addition there are two other fields, primarily used by an operating system, for saving and restoring selected VRs when swapping between jobs or other tasks.

In addition to the System/390 scalar instructions, there are 171 vector instructions, including

- Loads and stores—by stride, indirect, and masked. (Stride refers to the spacing of the elements of a vector

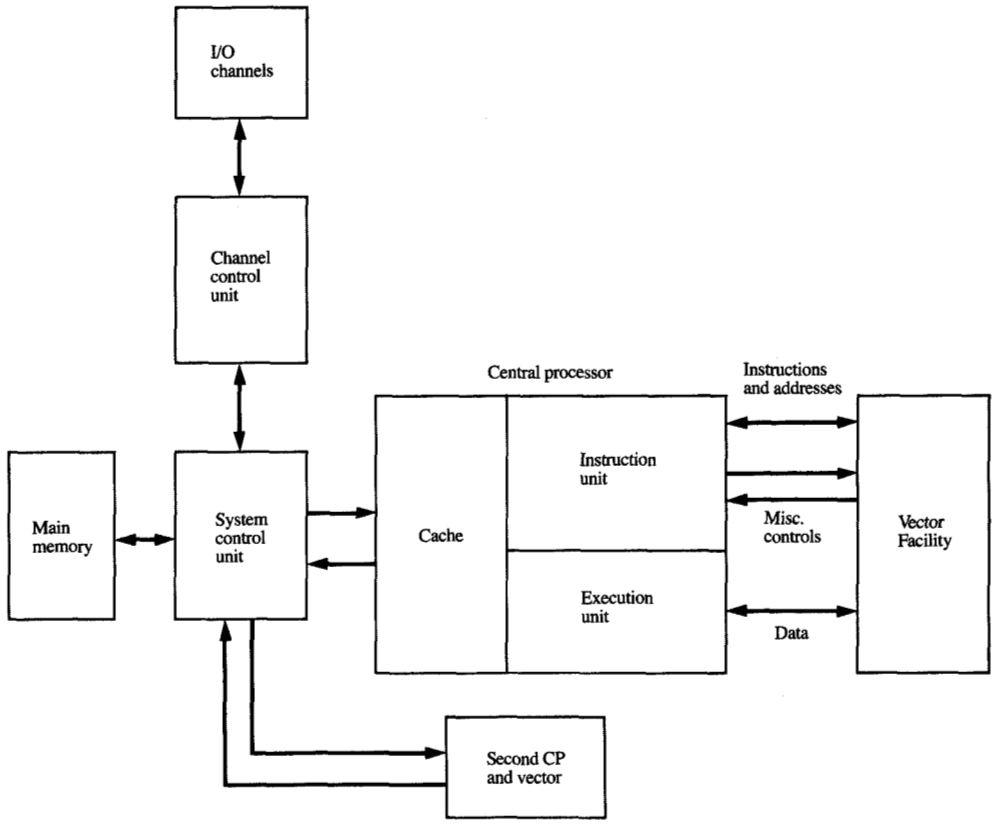


Figure 2

Vector Facility attached to the ES/9000 system.

in storage. Stride 1 means that elements are adjacent, a stride 2 vector is one in which there is a gap of one element width between the actual elements, and so forth.)

- Arithmetic/logical—including add, subtract, multiply, divide, AND, OR, XOR, accumulate, compare, minimum/maximum.
- Compound instructions—multiply and add/subtract, multiply and accumulate.
- Control-type instructions—load vector count, set vector mask mode, etc.
- Miscellaneous instructions—save changed VR, load element, etc.

Many vector instructions offer four variations: both operands from VRs (VV); one operand from a VR and the other from a scalar register (QV); one operand from a VR and the other from storage (VST); and one operand from a scalar register and the other from storage (QST). The scalar register may be either a general register or a floating-point register in the CP. In addition, many of the vector instructions operate on three types of data: binary integer, short floating-point (32 bits), or long floating-point (64 bits). Note that a VST compound-type instruction allows four operations to occur simultaneously: It fetches a vector from storage, multiplies it by a vector from a VR, adds that result to another vector from a VR, and puts the result back into a VR.

Vector facility operation

The vector facility hardware operates in conjunction with the central processor in executing vector instructions. In general, the CP initializes the instruction, the vector facility hardware takes over and runs it to completion, and

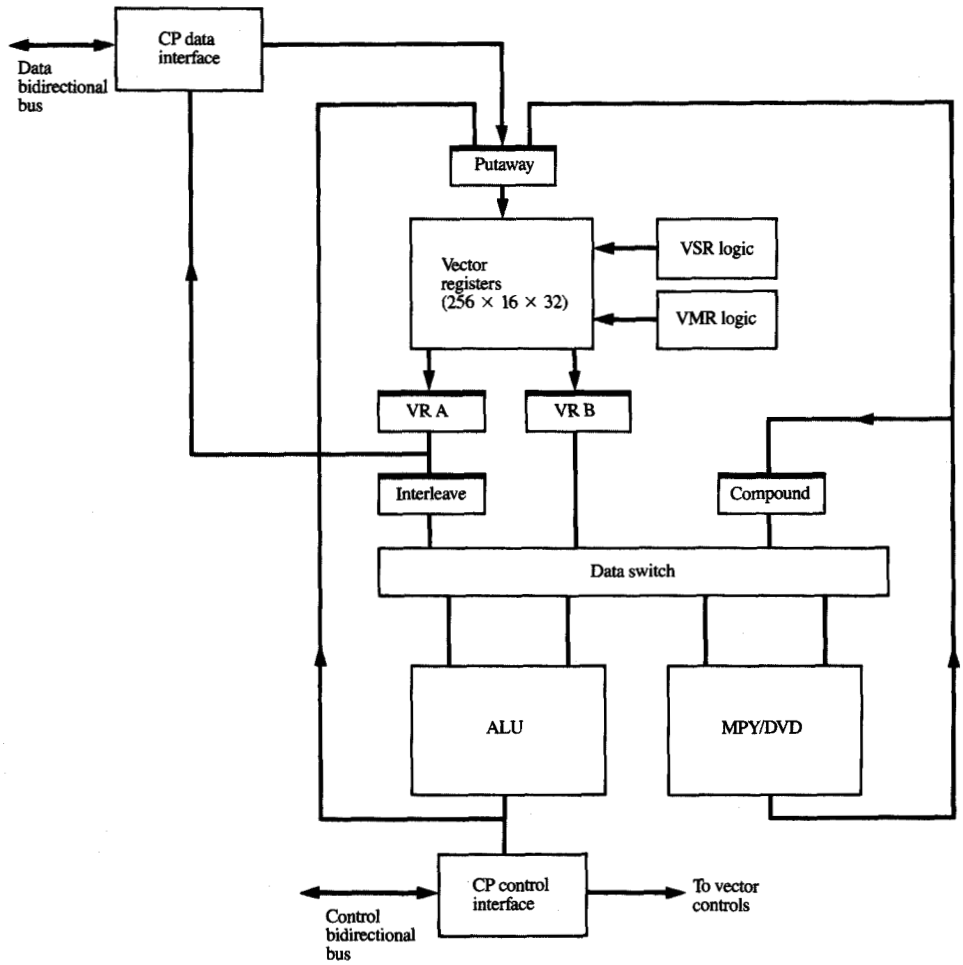


Figure 3

Vector data flow.

finally the CP performs any completion function necessary. **Figure 2** shows a high-level view of how the vector facility is attached to the rest of the ES/9000 system. Data go to and from the vector facility over a 64-bit bidirectional bus. Instructions come to the vector facility over a 32-bit bidirectional bus which is also used to send addresses to the CP for a few special instructions. In addition, there are numerous control lines to and from the vector facility which are not shown. One vector facility may be attached to each CP in an ES/9000 system.

This section discusses some of the principal hardware functional areas, including the vector registers, the arithmetic/logic unit, and the multiply/divide unit, and describes how vector instructions are executed. **Figure 3** shows a high-level data-flow diagram of the vector facility.

- *Vector registers*

The vector registers must be capable of reading two elements and writing a third during every machine cycle (for VV instructions). Because the array chips used are not capable of both reading and writing in the same cycle, the VRs are four-way interleaved. Elements 0, 4, 8, 12, $\cdots$, 252 of each VR are stored in the first interleave set, elements 1, 5, 9, 13, $\cdots$, 253 are stored in the second interleave set, and so forth. However, a VV instruction requires that the same element number from two different

VRs arrive at the arithmetic units during the same cycle; this also requires reading from the same interleave set at the same time, which is impossible. Therefore, the INTERLEAVE register (Figure 3) was added. This allows the read to be started for one vector operand one cycle before the other vector operand, and ensures that they both arrive at the arithmetic units in the same cycle.

An additional complication was that an element about to be written into the VRs must not arrive when that interleave set is being used for a read operation. This was solved by analyzing each instruction to determine whether an interleave conflict existed. The solution required adding some stages to the arithmetic pipelines (which would not otherwise have been necessary) to delay the operand about to be written until that interleave set was no longer busy. This has a minimal impact on performance except on very short vectors.

Vector register recovery scheme

To improve machine availability, one new requirement on the ES/9000 Type 9121 Vector Facility was that it must be possible to recover from hardware failures in the vector register arrays. Previous schemes used simple parity or traditional error-correcting codes (ECC). Parity checking does not allow for correction of errors; although ECC does allow recovery after an error, it is relatively difficult to implement with the vector facility chip partitioning, requires a significant amount of logic, and tends to impact the overall design of the vector facility.

To avoid these shortcomings, another recovery method was invented. It relies on current array technology, which provides much denser array chips than are needed in the vector register application. This extra density is used to provide a redundant copy of all the data stored in the VRs, for the purpose of error recovery. The method is used to recover from both transient and most types of permanent errors in the array chips.

Figure 4 shows a high-level view of the chip partitioning for the vector register arrays. This layout allows reading from two different interleaves and writing to a third interleave in the same cycle. Note that when a write operation is performed on a particular interleave set, a corresponding write operation is performed on the parity array interleave set. The recovery scheme involves adding another chip to hold the backup data. Note that this chip does not need to be interleaved like the primary arrays, since data are written only to a single primary interleave set during a cycle. During normal operation, data are never read from this backup array chip. Also, note that the backup chip holds four times as much data as the primary chips, since it must be able to contain all the data in the four different primary interleaves. This is possible because 256Kb array chips are used and only 128 Kb are needed for the backup chip.

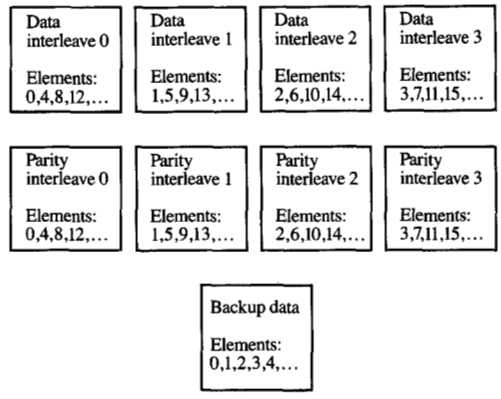


Figure 4

High-level partitioning of vector register arrays. This diagram has been simplified in that this entire layout of chips is actually duplicated. One set holds the even-numbered VRs and the other set holds the odd-numbered VRs.

In addition to the backup array for redundancy, as described above, another type of redundancy exists in the primary data and parity arrays. Since only a small fraction of the total array chip is actually used in the vector register application, a second address space within the same physical chip is made available via a scan-only latch [4]. During normal operation, only the first address space on the chips is used. However, after certain types of recovery are performed, the second address space may be used.

During normal operation, every time data are written into the primary arrays, the identical data are written into the corresponding location in the backup array chip. As data are read from the primary arrays, normal parity checking is done with the data stored in the parity array chips. **Figure 5** shows a diagram of logic involved in the recovery scheme. If a parity error is detected by the logic, the following recovery algorithm is used:

- The processor controller stops all processing being done by the system.
- The processor controller then reads out the failing element from the primary data arrays (D_p), parity arrays (P_p), and the corresponding data element in the backup arrays (D_b). These sets of data are then compared, with the following actions being taken:
  - If PARITY(D_p) = P_p, the problem was probably due to the error-checking logic itself.
  - If PARITY(D_b) = P_p, then D_p was bad, and the processor controller replaces D_p with D_b.

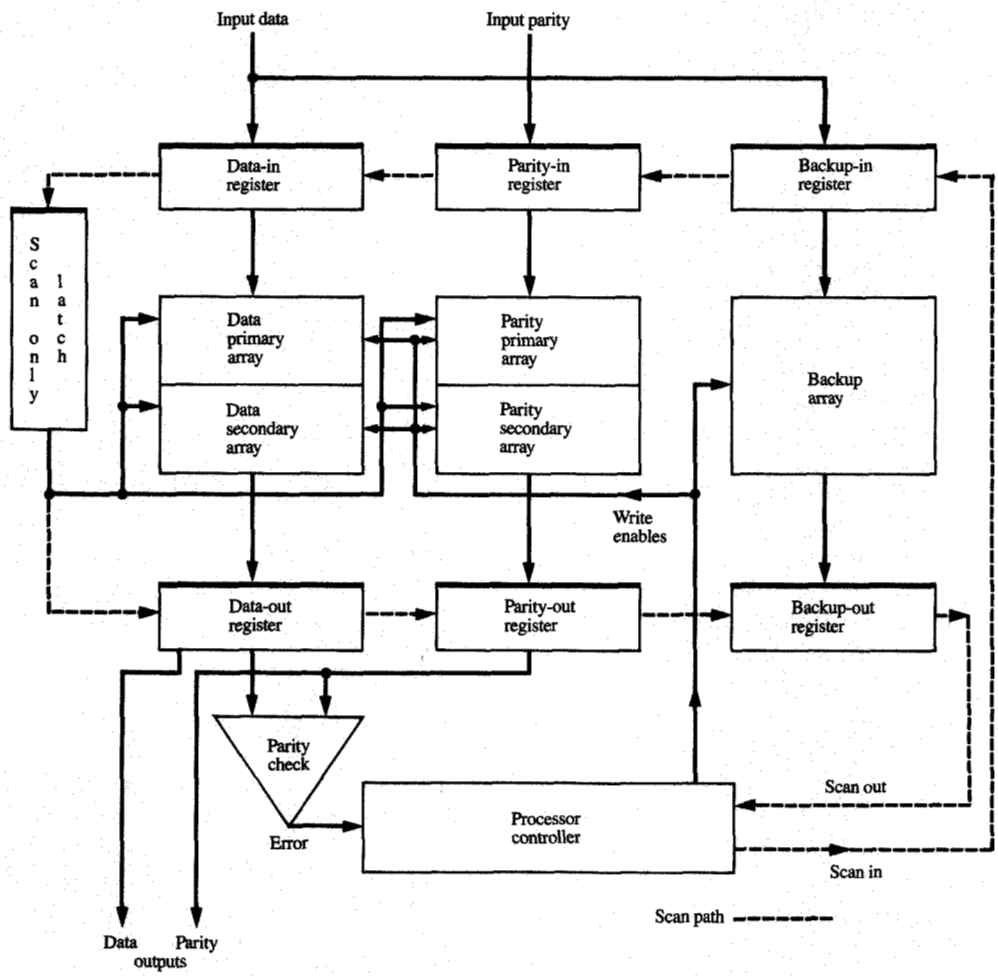


Figure 5

Vector registry recovery scheme data flow.

- If $D_b = D_p$, then P_p was bad, and the processor controller corrects the parity in the P_p element.
- If (PARITY(D_p) $\neq P_p$) AND (PARITY(D_b) $\neq P_p$), the failure is unrecoverable, because nothing matches.

- If the processor controller determines that the failure was one of the first three, the failing instruction is retried. If the retry is successful, operation continues normally.
- If the instruction retry is not successful after a certain number of attempts, the problem is probably a permanent array failure instead of a transient one. Since the primary data and parity arrays have two address spaces, with only one being used normally, if there is a permanent failure the processor controller can switch to the second address space by changing the value in the scan-only latch. The current job is aborted by the operating system and further processing continues normally. Note that no parity is kept on the data stored in the backup array, since it can be shown that there is no additional benefit, from an error-detection or error-recovery point of view, to be derived from including this extra hardware. The recovery algorithm checks to see that the backup data are consistent with the primary parity before replacement, which is in itself sufficient to ensure that these data are correct to a reasonable statistical certainty.

• *Vector arithmetic/logic unit*

The arithmetic/logic unit (ALU) is a pipelined design which executes addition and subtraction on binary integers and on short and long floating-point data. It also performs the logical operations AND, OR, and XOR. In addition, the ALU is used during compare, minimum/maximum, accumulate, shift, and certain load-type operations, and during the load and store indirect instructions to calculate the storage addresses. It consists of four major functional areas: preshift, adder/logical, postnormalization, and exponent.

The ALU uses several types of error checkers. Most of the checkers are traditional parity, redundancy, or parity-prediction checkers. Modulo 15 residue checking is used instead of the more traditional parity-prediction schemes for the main adder/logical section because duplicated carry-generation logic to achieve full error detection would not fit on one chip with the normal functional logic. This provides approximately 93.3% detection on any type of error within the adder/logical section, including multibit failures.

• *Vector multiply/divide unit*

The vector multiply/divide unit performs both single- and double-precision floating-point multiply and divide operations and integer multiply operations and occupies approximately one third of the vector facility TCM. It consists of a pipelined four-stage multiplier. For divide operations, parts of the multiply logic are used in conjunction with additional logic to create an iterative divider which produces four results in 16 cycles for double-precision divide operations and four results in ten cycles for single-precision divide operations. **Figure 6** shows a high-level diagram of this unit.

Multiply

For multiply operations, the unit is configured with four main parts: input registers, partial product generators, final product assimilation, and postnormalization. One input operand is first latched in the multiplicand registers and the other is latched in the multiplier registers. There is also one additional staging register for each operand that solves the vector register interleave conflict problem. If the operation is an integer multiply, the data must be shifted right eight bits to align with the place where the fraction would go if it were a floating-point operation. Note in the diagram that the multiplicand is simultaneously latched in all four MCAND_2 latches, which hold different values for divide operations. Before the multiplier is latched in the MPLER_2 registers, it is recoded using the traditional Booth algorithm [5].

The next stage of the multiplier consists of the four parallel partial product generators (each handling seven partial products), shown as partial product generators A, B, C, and D in the diagram. Each of these multiplexes multiples (+1, −1, +2, −2, and 0) of the multiplicand and performs a 7-to-2 carry-save add, followed by a chip-wide carry-propagate add.

The next section in the multiplier is the assimilate logic, which combines the four partial products from the partial product chips and does a 112-bit-wide carry-lookahead add to obtain the final product. Note that the carries from the chip-wide addition are also added in here, along with a 29th partial product from the most significant recode group. (Recoding a 56-bit floating-point fraction yields 29 recode groups. The leftmost group is added at the final stage.)

The final section of the multiplier performs the postnormalization function. The logic has the capability of selecting from either of two divide buses (within a single multiply/divide unit, quotients are generated in two different places) or the multiplier. Postnormalization (for multiplication) is done here along with an eight-bit left shift to put integer data back in its correct format. There is also another staging register, to solve the vector register interleave conflict problem.

Divide

For divide operations, the multiplier is configured as four subpipes, each operating semi-independently to produce a quotient. Both dividends and divisors enter the unit on the multiplicand bus, with the dividend preceding its divisor by one cycle and appropriate bit shifting being performed. At the same time, the first few bits of the divisor address a table which yields an approximate reciprocal of the divisor. (The table values are "stored" in combinational logic, logically 10 bits wide by 1024 deep.) After the start-up period, four distinct divides occur simultaneously, and the four divisors are stored in the MCAND_2 registers throughout the operation.

The following three equations show the calculations being performed to produce the quotient:

$$QDIGIT_i = RECIP \times RREM_{i-1}\,, \tag{1}$$

$$QSUM_i = QSUM_{i-1} + QDIGIT_i\,, \tag{2}$$

$$REM_i = REM_{i-1} - (QDIGIT_i \times DVSR), \tag{3}$$

where $REM_0 = DVND$, $QSUM_0 = 0$, and $RREM_i$ is an approximation to the full REM_i. The basic iteration consists of multiplying an approximate remainder (on the first iteration, the dividend is the remainder) by the approximate reciprocal obtained from the table lookup to produce a net of eight quotient bits [Equation (1)]. This is shown in Figure 6 as the QGEN block. This new quotient is then added to the previous partial quotient in the QSUMMER [Equation (2)]. While this addition is taking place, the incremental quotient is multiplied by the divisor and subtracted from the previous remainder to produce a

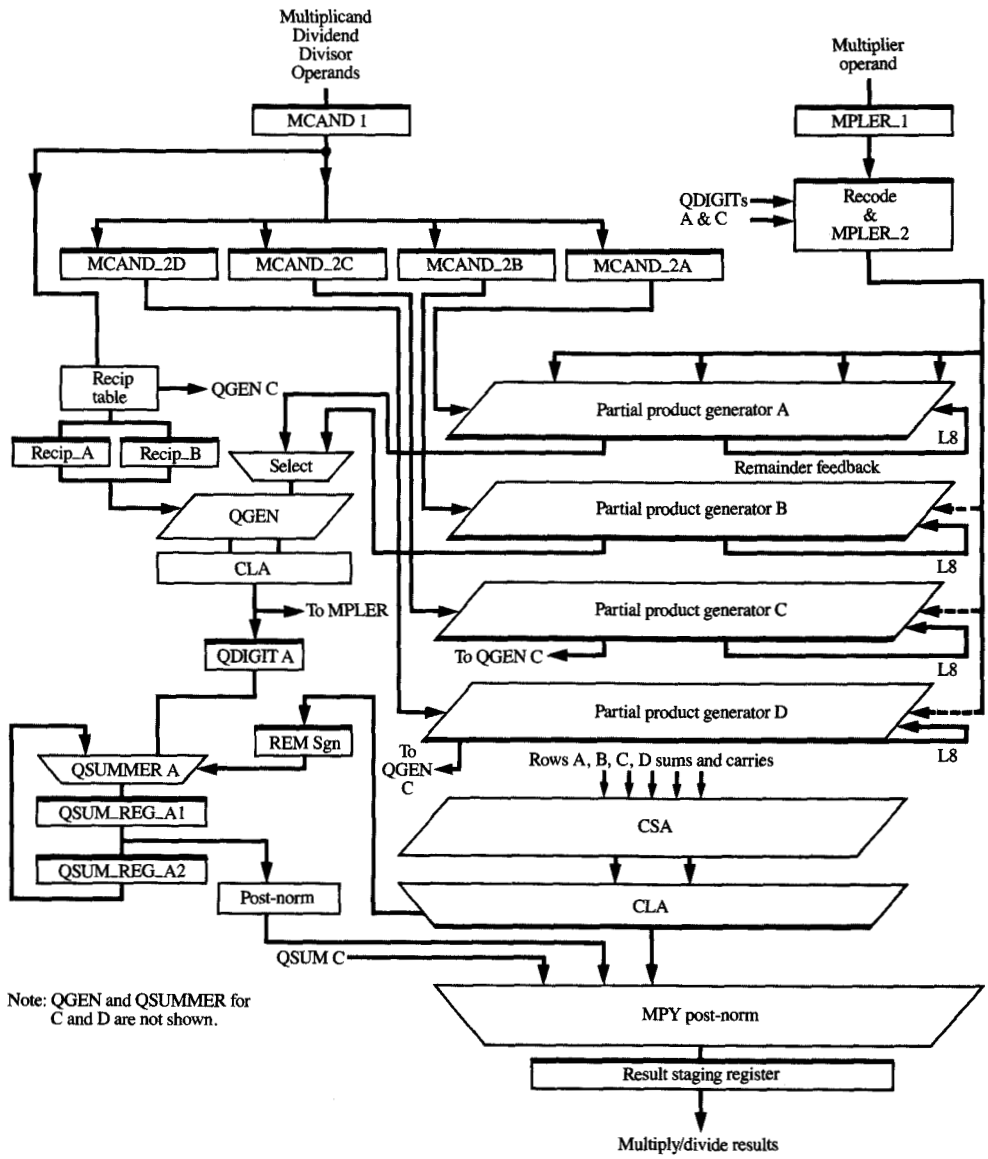


Figure 6

Diagram of multiply/divide unit.

new remainder [Equation (3)]. This process is repeated seven times. Since the remainder may be negative after the final iteration, a "1" is subtracted from the least significant bit of the quotient to yield a truncated 56-bit result. The quotient then undergoes postnormalization in the QSUMMER and is passed to the multiplier postnormalization logic to be placed on the output bus.

It should be noted that there are two different quotient generators and two different QSUMMERs. One pair operates on the data in partial product rows A and B and the other pair on partial product rows C and D. Each iteration is two cycles long and is staggered between partial product rows so that there are no conflicts. Therefore, at any given time four divide operations are executing.

Error checking

To ensure that correct results are being obtained from the unit, modulo 15 residue checking is performed. For

multiplication, the traditional residue-checking equation is used:

$$\text{Res}\,(PROD) \stackrel{?}{=} \text{Res}\,[\text{Res}\,(MCAND) \times \text{Res}\,(MPLER)]. \quad (4)$$

Note that this requires the residue to be taken on the entire 112-bit result. Also, integers that are negative must be accounted for by modifying their residues before the equation is checked. For division, the equation that is checked is

$$\text{Res}\,(DVND) \stackrel{?}{=} \text{Res}\,\{[\text{Res}\,(QUOT) \times \text{Res}\,(DVSR)] + \text{Res}\,(REM)\}. \quad (5)$$

Again, because the final remainder may be negative, its residue must be modified before the equation is checked.

Although the vast majority of the multiply/divide logic is checked by this single residue checker, there are several additional checkers. These are mainly redundancy checkers on the control logic and parity checkers on the input operands.

- *Vector facility controls and instruction execution*

The vector facility monitors all instructions as they are executed by the CP. When a vector instruction is decoded, the vector facility begins setting up to perform the instruction. Simultaneously, the CP sends the vector any scalar register data it needs. The VSR is kept in the CP and is also sent to the vector (if required). For VST and QST instructions, the CP initializes address generation logic within the cache to provide storage data automatically to the vector facility as quickly as possible.

At this point the vector hardware essentially runs the instruction to completion. When it is finished, it notifies the CP if it has any VSR or scalar register data to return; such data are then stored in the appropriate location within the CP. Finally, the instruction-processing logic within the CP is notified to begin executing the next instruction.

One of the more complex instructions is the STORE INDIRECT instruction. This requires the vector facility to generate storage addresses by adding a 32-bit integer vector in a VR to a base and displacement value from the CP in order to store another vector. Since this instruction can easily access a different page in storage for each element, some hardware was added to improve performance. A small table is kept which contains the logical address of the last three consecutive pages that have been accessed. When a page that is not in the table is about to be accessed, the vector facility requests the CP to test the page for an access exception. If none is found, the table is updated with the new address and processing continues. With this mechanism, the performance of the instruction is significantly improved, because it is no longer necessary to pretest each new page that is encountered.

- *Interface with the cache*

The ES/9000 Type 9121 Vector Facility is different from most vector processors in that storage data are accessed through the cache instead of directly from main memory. (Storage data physically pass through the CP instead of directly from the cache, but this is really due to packaging issues and does not affect the way in which data are logically accessed.) This scheme was selected because it allows reasonable performance for vectors with nonunit strides without incurring the added cost of a highly interleaved main memory. Of course, it also eliminates the problem of keeping cache data synchronized with main memory data, which would be a problem if the vector accessed main memory directly.

Generally, when the vector storage request hits in the cache, the data are returned immediately and the performance is excellent. However, when the vector storage request misses in the cache, the vector accesses storage data in three different modes, depending on the type of instruction and the stride, in an attempt to maintain the best performance possible. The first is cache-fill mode, which lets the cache automatically generate addresses and prefetch lines of data from main memory if they are not already in the cache. This permits the pipelining of multiple cache lines from main storage. It is used for almost all stride 1 and 2 instructions. The second method is cache-bypass mode, which is used in conjunction with cache-fill mode during stride 1 double-precision and stride 2 single-precision instructions. On cache misses, as data come from main memory, they are simultaneously stored in the cache and also routed directly to the vector facility. This is feasible, since for these strides and precisions the vector facility can accept data at the same rate that main memory can send data. This eliminates the need to both write and read the cache arrays, which would decrease performance. The final method is the normal access mode used for most scalar instructions, which is used for vector accesses in all other cases. Here the CP generates an address and the data are sent to the vector facility one cycle after being received into the cache from main memory. Note that the entire line must be transferred into the cache, even if only one element is contained within that line. This can result in degraded performance for large strides if the data are not used in some following instruction. Since it is not known at the start of an instruction whether the vector data are in the cache, the CP makes the decision on the mode at the beginning of the instruction, and the same mode is used throughout the execution of the instruction.

Vector performance evaluation

The preceding section described the design considerations and decisions for the ES/9000 Type 9121 Vector Facility. As the design progressed, each design choice was

monitored and evaluated for its potential impact on performance. This monitoring activity was based on modeling and analytic techniques and ensured that final results would be consistent with the objectives. Final evaluation relied on actual measurements. Since the design was an extension of the 3090 Vector Facility design, the performance evaluation scheme is similar to the one used to evaluate the 3090 vector facilities.

Vector facility performance was evaluated by using numerically intensive computing (NIC) application programs. NIC applications are characterized as follows:

- They make extensive use of floating-point computations, with large amounts of arithmetic data being operated on in loops for long periods of time. Generally, the floating-point content is greater than 50% of the instruction count and total job CPU time.
- They use FORTRAN as the primary language.
- They run at high CPU utilization rates, typically exceeding 90%.
- They can have significant I/O requirements.

NIC application programs were chosen as the framework for evaluation for several reasons. First, they allow the direct evaluation of the product design objectives on the basis of an extensive investigation of NIC application characteristics and the role of vector speed within this application performance context. Second, they provide the user with a more realistic view of potential performance improvements when migrating from scalar-only systems. The computationally intense nature of these NIC programs results from iterative mathematical computations on sets of data typically found in loops coded as FORTRAN DO statements. However, loop performance comparisons and kernel measurements are only part of the overall picture. By using application programs, the entire problem solution, complete with the I/O, data manipulation, and all the other noncomputational processing, can be evaluated, rather than just the more easily vectorized DO statements. Loop or kernel measurements should not be used to characterize the overall *system* performance achieved through the use of complete applications. Finally, by using a set of applications covering a variety of disciplines, users can get a reasonable estimate of the performance improvement they may realize from the Vector Facility system. Since actual performance varies by application program, a user can determine the performance for an application by selecting the stated performance for a specific job or set of jobs that closely matches the characteristics of the application of interest. As mentioned, one reason for differing gains in performance is the variation in vectorizable content. Because the ES/9000 Type 9121 Vector Facility accesses storage data through the cache, another factor affecting performance is the cache behavior of the application. A variety of factors in a customer application, such as stride and vector length [6], may influence cache performance. Therefore, if a set of application programs representative of a variety of relatively long production NIC runs is collected and the members are measured separately, users will be able to relate the results directly to their own NIC requirements.

With application performance established as the framework for the Vector Facility evaluation, a large set of application programs was collected and analyzed for suitability as benchmark applications. These programs came from a variety of sources, including

- Publicly available codes/applications.
- Programs used by customers as benchmarks.
- Customer production or research applications obtained under special agreements.
- Licensed application software packages, together with specific problem data or vendor-supplied benchmark data.

A subset of these applications, representing the types of production workloads expected to be run on the Vector Facility, was chosen, and became the Scientific/Engineering Application Program (SEAP) benchmark set. The job set was constructed to represent the engineering/scientific production environment, which consists of long-running applications as well as some of short duration. All contain a significant volume of floating-point computation. Groups of one to three jobs represent a specific discipline (again, each job within a group is measured separately). As the number of jobs included in the SEAP job set increased, they were partitioned into two groups. A second group labeled "special interest codes" was created which contained jobs that were based on segments of large applications or possessed an unusually high vector content. The SEAP job set has been used for the evaluation of all the IBM vector facility offerings, including the ES/9000 Type 9121 Vector Facility. **Table 1** contains the names of the applications within the SEAP job set, and the NIC discipline represented by each.

The SEAP set is not static; it varies as new application areas and advanced techniques are created. The advances are reviewed and studied for potential addition to the SEAP set. At the present time, two chemistry codes are being considered for inclusion in the job set. Several SEAP jobs are currently being reviewed for representativeness, and, if necessary, will be replaced with newer versions. Some of the application codes are continually being reviewed against the need to update the product level. Applications from disciplines not currently represented by the SEAP set are being considered for inclusion in the job set.

SEAP job descriptions

The following are descriptions containing the solution techniques and the mathematical algorithms used for each SEAP job [3].

Static structural analysis This is a finite element job designed to form and solve a large stiffness matrix of a static analysis problem with little I/O overhead and high processor utilization. The program spends much of its computational time in solving large sparse systems of simultaneous linear equations. Techniques for solving such systems are usually based on Gaussian elimination, for example L-U decomposition and frontal methods. Storage methods such as submatrix, compressed column, and skyline are used extensively.

Buckling analysis This finite element job performs a type of nonlinear analysis of a structure in which the stiffness matrix is used to calculate a deflection. Another stiffness matrix is formed, and the process is repeated until the buckling failure of the structure is determined.

Thin-layer fluid dynamics This job is a three-dimensional, thin-layer Navier–Stokes problem simulating the time-dependent behavior of steady or unsteady inviscid and viscous compressible flows around simple body configurations.

Crystal growth fluid flow This fluid flow application simulates the fluid flow in Czochralski crystal growth, in which a large-diameter cycle is produced by slowly extracting it from a crucible of melt.

Turbine blade analysis This application uses a finite difference algorithm to solve the three-dimensional air flow between the blades of a rotating turbine stage.

Seismic analysis This application recovers information about underground terrain from seismic time-trace data. The programs

- Apply one-dimensional moving average filters in either the time or frequency domain.
- Apply two-dimensional filters in the frequency-wave number domain (2D FFT).
- Perform normal move-out (NMO) correction and stacking (summing) of traces from common midpoint data, and long correlation in the frequency domain (FFT).
- Solve a seismic modeling problem based on a finite element model.

Black-oil-reservoir simulation This job is a three-phase black-oil-reservoir modeling program (BOAST, from the U.S. Department of Energy) which is used to simulate both the primary depletion and the secondary recovery operations in a two- or three-dimensional black-oil reservoir. The solution uses the Implicit Pressure, Explicit Saturation (IMPES) method to solve for the fluid flow in a reservoir. The pressure and saturation equations for the oil/gas/water systems are approximated using a finite difference method. The resulting system of linear equations is solved using an iterative Line Successive Over-Relaxation (LSOR) technique.

Table 1 SEAP job set.

SEAP applications	*Discipline*
Static structural analysis	Finite element design
Buckling analysis	Finite element design
Turbine blade analysis	3D air flow
Thin-layer fluid dynamics	3D compressible flow
Flow field analysis	Fluid dynamics
Large-eddy simulation	Fluid dynamics
Crystal growth fluid flow	Fluid flow simulation
Biomolecular dynamics	Computational chemistry
Time series	Financial analysis
Seismic analysis	2D seismic analysis
Black-oil-reservoir simulation	Reservoir modeling
Special interest codes	
Dense-matrix inversion (long)	Linear algebra
Dense-matrix inversion (short)	Linear algebra
Sparse-matrix decomposition	Linear algebra
Quantum chromodynamics	Quantum physics

Dense-matrix inversion This job is a segment of a customer application benchmark that computes the inversion of a dense matrix. The SEAP set contains both a long- and short- (double- and single-) precision version of this benchmark.

Biomolecular dynamics This application program performs simulations of the molecular dynamics of water and counter ions around nucleic acids. The Newton–Euler equations of motion are solved by using 5/6-order predictor-corrector methods to evaluate the molecular positions and molecular velocities needed to understand the hydration behavior.

Time series In this job, econometric analysis is performed by the Wisconsin Multiple Time Series Package, using the auto-regressive, integrated moving average (ARIMA) process to determine the parameters of an econometric model, estimate the sensitivity of the model, and forecast econometric parameters.

Flow field analysis This job uses a finite volume method to analyze the three-dimensional inviscid transonic flow

Table 2 SEAP job characteristics.

SEAP application	*MFLOP count*	*Scalar FP instructions* (%)	*Scalar FP operations* (%)	*EXCP count*	*Total virtual storage* (MB)	*Percent vector*
Static structural analysis	13672	75	37	26505	14.8	83
Buckling analysis	6262	64	32	42632	14.8	74
Turbine blade analysis*	7887	76	34	116	32.8	93
Thin-layer fluid dynamics*	9289	86	42	120	12.0	91
Flow field analysis	22989	83	41	209	27.8	92
Large-eddy simulation*	4077	92	36	107	5.5	95
Crystal growth fluid flow	790	69	40	564	1.4	84
Biomolecular dynamics	1857	60	31	1784	2.6	68
Time series analysis	418	47	26	1059	9.1	69
Seismic analysis*	1488	64	33	6234	1.6	93
Black-oil-reservoir simulation	4559	45	21	223	5.5	83
Special interest codes						
Dense-matrix inversion	2012	83	33	20	8.1	98
Dense-matrix inversion*	2012	83	33	20	4.1	98
Sparse-matrix decomposition	1360	72	47	20	13.5	88
Quantum chromodynamics	1783	72	38	58	6.1	99

*Indicates short precision.

about an arbitrary wing/fuselage/tail/fin configuration by solving the unsteady Euler equations. Adaptive dissipation is used to capture the shock waves. A multigrid algorithm accelerates convergence to a steady state.

Large-eddy simulation This application simulates homogeneous fluid flow in large eddies. Taylor's rapid distortion theory is used to model the homogeneous turbulence. Navier–Stokes equations, including viscous terms, are solved by means of a fourth-order Runge–Kutta algorithm. The coefficients of a truncated 3D Fourier series represent the turbulence field.

Quantum chromodynamics This is a university code from quantum physics research which performs a quantum chromodynamics simulation of interactions between elementary particles. Interactions between quarks and gluons are simulated by updating the gluon field and permitting properties of the quark and gluon fields to be measured. An algorithm based on stochastic quantization via the Langevin equation is used to update the gluon field.

Sparse-matrix decomposition This job is a segment of a customer application benchmark that computes the Cholesky decomposition of a sparse matrix.

Table 2 presents some of the SEAP job characteristics. *MFLOP count* is the total number of floating-point operations in the program, in millions (excluding floating-point load and store instructions). *Scalar FP instructions* is the percentage of the total instructions in the scalar job which are floating-point instructions, including loads and stores. *Scalar FP operations* is the same, but includes only computational floating-point instructions (no loads or stores). *EXCP count* is a measure of the I/O operations (reads and writes) for each program. *Total virtual storage* is an indication of the number of megabytes (MB) of storage used by each job. *Percent vector* is the percentage of the scalar application CPU busy time that was vectorized [3].

The values given in Table 2 were determined when the jobs were initially executed in scalar mode. Modifying the applications to use the Vector Facility can cause these values to change; so can changes in the compiler. These values should be used as approximations of the floating-point content of the jobs. See [3] for specific details. It should be noted that slightly over half the applications in the SEAP job set are above the mid-range of vectorizability. In selecting applications for SEAP, several criteria are employed. The major priority is to find applications which represent the type of work a prospective IBM Vector Facility system user would run. It does not necessarily follow that this NIC application will also fall into the mid-range. Additionally, some of the SEAP jobs have been a part of the job set for over five years; it should be noted that there are several factors which tend to push the vectorization extent upward, such as VS FORTRAN compiler enhancements, improved algorithm techniques, and additional Engineering/Scientific Subroutine Library (ESSL) [7] function and enhancements. As a result, although the design objectives have remained the same, the SEAP characteristics have changed. It should be noted that the Vector Facility performance is good well into the high vectorizability range, especially if the application can make efficient usage of the cache storage hierarchy through the techniques listed previously.

Table 3 CPU busy seconds.

SEAP applications	*ES/4381 Model 91E scalar*	*ES/9000 Model 320 scalar*	*ES/9000 Model 320 vector*	*ES/3090 Model 180J scalar*	*ES/3090 Model 180J vector*	*ES/9000 Model 480 scalar*	*ES/9000 Model 480 vector*
Static structural analysis	10853	1702	754	1340	646	1714	827
Buckling analysis	5356	869	429	703	375		
Turbine blade analysis	8176	1420	463	960	415		
Thin-layer fluid dynamics	7504	1218	478	928	423	1231	507
Flow field analysis	17456	2674	1054	2148	936		
Large-eddy simulation	3571	554	158	455	129		
Crystal growth fluid flow	535	91	39	71	35	91	39
Biomolecular dynamics	1752	289	142	253	127		
Time series analysis	374	60	25	54	24.6		
Seismic analysis	1276	214	52	164	45		
Black-oil-reservoir simulation	5578	953	348	873	325	953	352
SEAP special interest codes							
Dense-matrix inversion (long)	1874	255	80	68	256	94	
Dense-matrix inversion	2082	253	67	61	254	68	
Sparse-matrix decomposition	1006	140	49	46			
Quantum chromodynamics	1326	224	39	37			

The SEAP job set has proved to be a valuable metric for engineering/scientific applications; it has been used to evaluate vector facility performance for the ES/3090 family and the recently announced ES/9000 processors. SEAP is measured and reported in both scalar and vector mode, for single jobs on one processor and multiple copies on multiprocessor configurations. The internal throughput rate ratios have been especially useful for evaluating various migration path options. SEAP has provided the system performance data required and set the expectation levels necessary for customer performance evaluations.

Measurement methodology and results

The SEAP job set was run on all IBM ES/9000 Type 9121 machines. The SEAP job run times for Models 320 and 480 (**Table 3**) demonstrate the hardware performance improvement available to IBM ES/4381™ users through the extension of the 3090 vector design to the air-cooled environment. Performance numbers for the ES/3090 Model 180J with Vector Facility are also provided, so that the performance of the new machine can be compared to that of an existing 3090 machine. Software levels were maintained for all measurements and are listed as follows:

- MVS/ESA™ Version 3, Release 1.3.
- VS FORTRAN Version 2, Release 4.
- ESSL Version 1, Release 4.0.
- MSC/NASTRAN® Version 64D.

- *Methodology*

The measurements were done in two different modes:

1. *Single initiator* This mode is used for uniprocessor comparisons. Each job was run on a 9121 Model 320 processor with only one active initiator. Both scalar and vector SEAP programs were measured, and CPU busy time recorded (SRB + TCB).
2. *Multiple initiator* This mode is used to show the capacity of the 9121 Model 480. Two copies of a subset of the SEAP jobs were run. Both scalar and vector versions of the subset were run. To ensure simultaneous processing at high utilizations by each central processor, four copies of the job for each CPU were submitted, and the queue held. The number of active initiators was 2 for the 480. All jobs were released from the queue at the same time, and the average CPU time for the jobs executed during the measurement period was used for the internal throughput rate comparisons. A few jobs required additional steps to attain the highest possible central processor utilizations during the measurement interval. Additional details are available in [3].

The SEAP job set was compiled using the highest level of optimization [OPT = 3 for scalar and VECTOR(LEVEL (2)) for vector measurements] and the VECTOR(SIZE = (ANY)) option. This last option is recommended for producing applications independent of the model-dependent vector parameters.

Jobs were run in an unconstrained environment (e.g., sufficient DASD and I/O paths, paging data sets, and real storage) to minimize performance bottlenecks.

- *Results*

Table 3 contains the measurement results. The CPU busy times for both vector and scalar processors are shown. Performance data are shown only for the multi-initiator subset on the Model 480. The data show a substantial

Table 4 Internal throughput rate ratios.

SEAP applications	*ES/9000 Model 320 scalar to ES/4381 Model 91E*	*ES/9000 Model 320 vector to ES/4381 Model 91E*	*ES/9000 Model 320 vector/scalar ratio*	*ES/9000 Model 480 scalar to ES/4381 Model 91E*	*ES/9000 Model 480 vector to ES/4381 Model 91E*
Static structural analysis	6.4	14.4	2.3	12.7	26.2
Buckling analysis	6.2	12.5	2.0		
Turbine blade analysis	5.8	17.7	3.1		
Thin-layer fluid dynamics	6.2	15.7	2.5	12.2	29.6
Flow field analysis	6.5	16.6	2.5		
Large-eddy simulation	6.4	22.6	3.5		
Crystal growth fluid flow	5.9	13.7	2.3	11.8	27.4
Biomolecular dynamics	6.1	12.3	2.0		
Time series analysis	6.2	15	2.4		
Seismic analysis	6.0	24.5	4.1		
Black-oil-reservoir simulation	5.7	16	2.7	11.7	31.7
SEAP special interest codes					
Dense-matrix inversion (long)	7.3	23.4	3.2	14.6	39.9
Dense-matrix inversion	8.2	31.1	3.8	16.4	61
Sparse-matrix decomposition	7.2	20.50	2.9		
Quantum chromodynamics	5.9	34	5.7		

improvement in performance for ES/9000 processors compared to the IBM ES/4381 Model 91E processor. Although the CPU busy times shown for Models 320 and 480 are essentially the same, the 480 was completing approximately twice as many jobs because it has two processors. This increase in throughput is reflected in the internal throughput rate ratios in **Table 4**. The data also indicate that although absolute vector performance for the Model 320 is a little less than that for the ES/3090 Model 180J, vector-to-scalar improvements comparable to the performance of the 3090 Model 180J were achieved.

There are two key reasons for the improvement in scalar CPU busy times shown for the ES/9000 processors compared to the ES/4381 Model 91E processor: reduction of machine cycle time from 52 to 15 ns and the presence of pipelining in the ES/9000 processors. Pipelining increases performance by allowing overlap of instruction execution phases. Other factors include a larger cache and a separate I/O processor on the ES/9000 processors. Instruction implementation differences between the two machines account for some positive and negative influences on the total amount of performance improvement. In comparing the performance of the ES/9000 machines to that of the Model 180J, it should be noted that the 180J has a 14.5-ns cycle time, a larger cache, and some instruction execution time differences which account for the increased performance. The performance improvement also depends on the characteristics of each SEAP job, and no single performance number can be used to reflect performance across the represented range of jobs. Table 4 contains the performance ratios of the two ES/9000 processors compared to the ES/4381 Model 91E, demonstrating the range of performance improvement that is attributed to the stated differences between the machines. Similar comparisons can be made to the ES/3090 Model 180J with the data provided in Table 3. The vector-to-scalar ratios indicate that the previously mentioned design objectives were achieved.

Summary

The ES/9000 Type 9121 Vector Facility system was designed to provide a balanced system implementation offering integrated scalar, vector, and parallel processing opportunities. The design is based on the architecture established in the 3090 family of vector facilities. A new chip and module packaging technology has allowed implementation in a single air-cooled TCM package. Reliability has been further enhanced, and a new vector register error recovery technique implemented.

Performance measurements were necessary to validate the vector performance of the new air-cooled processor design. Since the vector architecture is designed to provide excellent performance on numerically intensive computing (NIC) application programs, rather than specific loops or kernels, the SEAP job set is used for performance evaluation. This job set was a by-product of the 3090 Vector Facility design process and currently contains 15 application jobs. The SEAP job set was developed in support of the observation that performance in the NIC environment is application-dependent. SEAP has been used successfully to set the expectation levels for specific customer NIC environments.

The measurement results indicate a successful transfer of the 3090 Vector Facility function and performance to the ES/9000 Type 9121 environment.

Acknowledgments

We would first like to acknowledge all of the vector facility designers who worked on this project. Their dedication and attention to detail helped make this one of the most error-free designs ever produced. We would also like to thank B. Bartley, L. Boelhouwer, B. Cesare, R. Fuller, R. Grant, R. Larson, D. Ludovici, P. Marro, C. Mierzwa, W. Rockefeller, G. Salyer, B. Weiler, and T. Wilson for their contributions and comments on this paper.

References and notes

1. *IBM Enterprise Systems Architecture/370 and System/370 Vector Operations*, Order No. SA22-7125; available through IBM branch offices.
2. *IBM Syst. J.* **25**, No. 1 (1986): D. H. Gibson, D. W. Rain, and H. F. Walsh, "Engineering and Scientific Processing on the IBM 3090," pp. 36–50; W. Buchholz, "The IBM System/370 Vector Architecture," pp. 51–62; and R. S. Clark and T. L. Wilson, "Vector System Performance of the IBM 3090," pp. 63–82.
3. *IBM ES/3090 Engineering/Scientific Performance*, Technical Bulletin No. GG66-0245-03, IBM Washington Systems Center, Washington, DC, December 1989.
4. This type of latch may be written to or read in a serial manner by the processor controller when the vector clocks are stopped, using the level-sensitive scan design (LSSD) mechanism. For more information see E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the 14th Design Automation Conference*, New Orleans, June 1977, pp. 462–468.
5. O. L. MacSorley, "High-Speed Arithmetic in Binary Computers," *Proc. IRE* **49**, 67–91 (January 1961).
6. For engineers and scientists who want to improve performance by exploiting the IBM Vector Facility, the following are excellent references: David B. Soll, *Vectorization and Vector Migration Techniques*, Technical Bulletin No. SR20-4966-00, June 1986; and *Designing and Writing FORTRAN Programs for Vector and Parallel Processing*, Order No. SC23-0337-00, November 1986; both available through IBM branch offices.
7. The Engineering/Scientific Subroutine Library is a set of highly tuned NIC subroutines. ESSL Release 1.3 also introduced parallel functions. For more information, refer to the *ESSL General Information Manual*, Order No. GC23-0182; available through IBM branch offices.

Received October 9, 1990

Timothy J. Slegel *IBM Data Systems Division, Kingston, New York 12580.* Mr. Slegel is an Advisory Engineer in the Processor Systems Development group at IBM Kingston working on the design of future vector facility products. He received the B.S. and M.S. degrees in electrical engineering from Lehigh University, Bethlehem, Pennsylvania, in 1980 and 1982, respectively. After joining IBM in 1982, he worked on the arithmetic unit design of the 3090 vector facilities. In 1989 he received the IBM Outstanding Technical Achievement Award for his work on the IBM 3090 Model S Vector Facility. Mr. Slegel is a member of the Institute of Electrical and Electronics Engineers, Eta Kappa Nu, and Tau Beta Pi.

Robert J. Veracca *IBM Data Systems Division, Kingston, New York 12580.* Mr. Veracca is an Advisory Engineer in the Systems Performance group working on performance evaluation of the ES/9000 Type 9121 processors. He received a B.S.E.E. degree from Rutgers in 1959. He joined IBM in 1964, working on memory test equipment design and advanced manufacturing techniques in both technical and managerial capacities. In 1975 Mr. Veracca joined the Cryptographic Product design team and was involved in logic design and product engineering of the IBM 3845/46 processors. Various assignments in the Architecture and Standards group followed and were focused on 8100 System architecture and control-unit-to-terminal interface architectures.