Effective utilization of IBM 3090 large virtual storage in the numerically intensive computations of *ab initio* molecular orbitals

by M. Sakaki
H. Samukawa
N. Honjou

A new level of storage hierarchy, called Expanded Storage and available on the IBM 3090 system, is utilized by the MVS/XA™ operating system as high-speed paging equipment, allowing a user to hold application data in large virtual storage. To exploit the large virtual storage capability of the IBM 3090, a new application technique was developed for numerically intensive computations of* ab initio *molecular orbitals where high-speed transfer of a vast amount of intermediate data is a common requirement of most application programs. An application program running under MVS/XA was modified so that it could handle a vast amount of intermediate data in large virtual storage combined with Expanded Storage, achieving a 4- to 10-fold improvement in turnaround time at a CPU rate-determining step (SCF step) in medium-sized molecules.

The IBM 3090 computer system utilizes physical storage in conjunction with the virtual addressing architecture of System/370 Extended Architecture to permit direct addressing of a virtual memory of 2048 million bytes, or two gigabytes (GB), with a hierarchy consisting of high-speed buffer, central storage, Expanded Storage, and a direct-access storage device (DASD).[1,2] With the introduction of the IBM 3090, a new technology called Expanded Storage has been made available[1] and is consistent with the large memory requirements called for in engineering and scientific fields. In this paper, we describe a practical application of large virtual storage combined with Expanded Storage in the numerically intensive computation of *ab initio* molecular orbitals (MO), which is a powerful tool for solving various chemical problems.

High-speed transfer of a vast amount of intermediate data is a common requirement of most *ab initio* MO software programs, including the GAUSSIAN-82 program which was developed by the quantum chemistry group at Carnegie Mellon University.[3,4] We chose GAUSSIAN-82 as the sample application software for our present study, because it is not only a good prototype of *ab initio* MO software but also has been widely used for practical *ab initio* molecular quantum mechanics calculations (among the functions in GAUSSIAN-82, *ab initio* MO computation is the most fundamental one). Analysis of the original version of GAUSSIAN-82 [IBM Multiple Virtual Storage

(MVS) version[3]] disclosed that *ab initio* MO computations consumed a fairly large amount of elapsed time (turnaround time) compared with CPU time. Our experiments showed that the large elapsed/CPU time ratio was mainly due to the disk access bottleneck resulting from a large amount of intermediate data (two-electron repulsion integrals, or TEIs) in particular parts of the program. We assumed that the bottleneck could be reduced by holding the TEI data in a large virtual storage combined with Expanded Storage. Using this idea, we implemented a modification to the original program so as to keep as many TEIs as possible in a large virtual storage area combined with Expanded Storage, and to spill over the remaining TEIs into disk storage.

The HFR method is one approach widely used to obtain *ab initio* MOs.

Although Multiple Virtual Storage/Extended Architecture (MVS/XA™) allows up to a 2-GB Dynamic Common (DC) storage size, we selected a combination of appropriate DC size and disk spill file to achieve optimized system performance. Optimized DC size is selected on the basis of the sum of central storage and Expanded Storage, and multijob environment. Since the I/O routine of the TEIs (NTRAN) was coded in the 24-bit addressing mode (Assembler routine using the basic storage access method) in the original GAUSSIAN-82 program, we needed to add a mode-transformation function to the program for 31-bit addressing in the DC region above the 16-megabyte (MB) line, and to add a function for efficient treatment of spilled-over TEIs which were controlled below the 16-megabyte line. These additions resulted in a noticeable improvement in the turnaround time of the program, demonstrating the power of large virtual storage when combined with Expanded Storage.

Numerically intensive computations of *ab initio* molecular orbitals

The basic concept of the Hund–Mulliken, or molecular orbital (MO), method is to find approximate electronic wavefunctions for a molecule by assigning to each electron a one-electron wavefunction which in general extends over the whole molecule.[5] The computation of *ab initio* MOs conforms to the method that is based on the model derived from basic principles of quantum mechanics. The Hartree–Fock–Roothaan (HFR) method for the closed-shell system, which is made up of complete electron shells, is one of the approaches that have been widely used to obtain *ab initio* MOs. The advances in theory, algorithms, software programs, and computers have been extending its application area. Today, the HFR method can successfully address problems of practical importance in a variety of subjects ranging from chemistry to biology.[4,6–11] For extensive applications of the computations—larger in molecular size and more accurate in computational results—numerically intensive computations are requisite. Effective utilization of an advanced computer such as the IBM 3090 is important for the higher processing speed. Preliminary knowledge of *ab initio* MO computations is helpful for understanding the substance of application software and also the modification concept concerning the adaptation of GAUSSIAN-82 to the large virtual storage of the IBM 3090. In the following subsection, we outline the technical foundations of the *ab initio* MO computations.[4,5]

***Ab initio* molecular orbital computations.** We consider a molecule which is made up of M nuclei and N electrons. In the closed-shell HFR method,[5] MOs $\phi_i(x_\mu, y_\mu, z_\mu) \equiv \phi_i(\mu)$, in which each MO is a function of the Cartesian coordinates of the μth electron, are expanded by a finite set of l basis functions $\chi_p(x_\mu, y_\mu, z_\mu) \equiv \chi_p(\mu)$:

$$\phi_i(\mu) = \sum_{p=1}^{l} \chi_p(\mu) C_{pi}.$$

[This is a Linear Combination of Atomic Orbital (LCAO) approximation.]

As a set of basis functions, contracted Gaussians have often been employed in practical computations. A contracted Gaussian is a linear combination of Gaussian-type atomic functions (primitive Gaussian functions) taking the form of $x^J y^K z^L \times \exp(-\zeta r^2)$. The LCAO MOs are assumed to form an orthonormal set:

$$\int \phi_i^*(1)\phi_j(1)\, dv_1 = \sum_{p=1}^{l} \sum_{q=1}^{l} C_{pi}^* S_{pq} C_{qj} = \begin{cases} 1 & (i = j), \\ 0 & (i \neq j). \end{cases} \quad (1)$$

Here, dv_1 denotes the one-electron volume element without spin for the electron 1, the asterisk denotes complex conjugation, and $S_{pq} = \int \chi_p^*(1)\chi_q(1)\, dv_1$ is the overlap integral. We give each electron a wavefunction (called molecular spinorbital) which, in addition to the space coordinates, also contains the spin coordinates of that electron. A molecular spinorbital is represented by the product of an MO and an electron spin function. The total N-electron wavefunction appropriate for the description of an N-electron system should satisfy the antisymmetry principle; i.e., it changes only by a factor of -1 under an interchange of all the coordinates (including spin coordinates) of any two electrons. The total N-electron wavefunction is built up as a determinant of N molecular spinorbitals so as to satisfy the antisymmetry principle (such a determinantal wavefunction is often referred to as a Slater determinant).

The expansion coefficients $\{C_{pi}\}$ are determined by a variational method in which the total energy of the system associated with the approximated total N-electron wavefunction is minimized subject to the orthonormality conditions of the MOs, as in Equation 1. The resultant Euler equation is the Hartree–Fock–Roothaan (HFR) equation. The HFR equation for the ith MO $[(i = 1, 2, \cdots, (N/2)]$ is described as the following:

$$\sum_{q=1}^{l} (F_{pq} - \epsilon_i S_{pq}) C_{qi} = 0 \quad \text{for} \quad p = 1, 2, \cdots, l. \quad (2)$$

Here,

$$F_{pq} = \int \chi_p^*(1)\hat{h}_1\chi_q(1)\, dv_1 + \sum_{r=1}^{l}\sum_{s=1}^{l} [2(pq \mid rs) - (ps \mid rq)] \left(\sum_{j=1}^{N/2} C_{rj}^* C_{sj} \right)$$

are the elements of the $l \times l$ matrix $\mathbf{F}$, and ϵ_i is the one-electron orbital energy of molecular orbital ϕ_i.

$$\hat{h}_1 = -\frac{h^2}{8\pi^2 m}\Delta_\mu - \sum_{I=1}^{M} \frac{Z_I e^2}{r_{I_\mu}}$$

is the one-electron operator corresponding to electronic kinetic energy and electron-nuclear attraction, and

$$(pq \mid rs) = \int\int \chi_p^*(1)\chi_q(1) \frac{e^2}{r_{12}} \chi_r^*(2)\chi_s(2)\, dv_1\, dv_2$$

is called the two-electron repulsion integral (TEI). The r_{12} denotes the distance between electron 1 and electron 2. The order of the total number of these integrals is $l^4/8$. Thus, if larger basis sets (larger l) are employed, more intensive computations are required.

Since F_{pq}s are built up with MO coefficients $\{C_{pi}\}$, the solution necessarily involves an iterative process.[4] This technique is frequently called the *self-consistent-field* (SCF) method.[5] In the SCF procedure, we (1) assume a set of $\{C_{pi}\}$, (2) calculate the matrix $\mathbf{F}$, (3) solve Equation 2, and (4) compare the resulting ϕ_i with the assumed ones. Guided by this comparison, we choose a new set of $\{C_{pi}\}$ and repeat the outlined procedure until the assumed and calculated $\{C_{pi}\}$ agree[5] (SCF iteration). For the case employing a larger basis set (larger l), the SCF iteration requires more intensive computing.

Extensive efforts have been made in developing practical methods and computer software, including the HFR and more sophisticated methods.[4,6–11] These efforts have contributed in many significant ways to substantial innovations in the scientific field and to progress in scientific/engineering applications. In the following sections, we describe our exploration in adapting GAUSSIAN-82[3–4] to the large virtual storage of the IBM 3090.

Analysis of the original code for further improvement

Execution analysis. During experiments with molecular orbital computations using the GAUSSIAN-82 program (IBM/MVS version) distributed by Carnegie Mellon University,[3] we noticed that the elapsed time consumed by the computations was fairly large compared with the CPU time. To find bottlenecks, an analysis was performed for the closed-shell HFR computations[5] with the 3-21G basis set[4] on four medium-sized molecules, e.g., 1-methylbutadiene (C_5H_8), phenol (C_6H_6O), cytosine ($C_4H_5N_3O$), and guanine ($C_5H_5N_5O$), using the Execution Analyzer[12] and user timer. The system environment for the analysis was VS/FORTRAN Version 2.1, MVS/XA Version 2.1.3, and a 3090-200 with two Vector Facilities. The outline of the computations[13] is given in Table 1, and the timing data obtained are shown in Table 2. The timing data were obtained under the condition of an exclusive use of the resources of the whole system.

Finding a bottleneck. We selected an index, *Total Elapsed Time/Total CPU Time*, to determine whether a program is CPU-bound or I/O-bound. When the index is close to 1.0, the program is CPU-bound. The Total Elapsed Time/Total CPU Time ratio amounts to 3.0 to 7.6 in the computations. This implies that molecular orbital computations with the original program code are I/O-bound. The

Table 1 Outline of computations

Molecule	1-methyl butadiene	phenol	cytosine	guanine
Stoichiometry	C_5H_8	C_6H_6O	$C_4H_5N_3O$	$C_5H_5N_5O$
Number of electrons	38	50	58	78
Number of primitive Gaussian functions	99	123	135	180
Number of basis functions	61	75	82	109
Number of two-electron repulsion integrals	991 174	2 213 389	3 012 060	8 241 467
Number of SCF iterations	22	25	27	40

Table 2 Execution analysis of the original code (unit = second)

Molecule	1-methyl butadiene	phenol	cytosine	guanine
Stoichiometry	C_5H_8	C_6H_6O	$C_4H_5N_3O$	$C_5H_5N_5O$
CPU Time (s)				
TEI step*	37.32	75.12	96.12	257.66
SCF step†	25.82	60.62	87.35	336.20
Total	68.40	143.54	193.25	613.41
Elapsed Time (s)				
TEI step	38.71	76.59	97.83	301.56
SCF step	135.36	315.42	1067.68	4313.21
Total	215.10	433.79	1210.15	4671.25
Ratio of Total Elapsed Time to Total CPU Time	3.1	3.0	6.3	7.6
Ratio of Elapsed Time to CPU Time				
TEI step	1.0	1.0	1.0	1.2
SCF step	5.2	5.2	12.2	12.8
Weight percent of Elapsed Time				
TEI step	18	18	8	7
SCF step	63	73	88	92
Sum	81	91	96	99
Number of I/Os‡				
INT file	11 661	29 458	43 148	172 856

* TEI step denotes the computation of two-electron repulsion integrals.

† SCF step denotes the computation of the Hartree–Fock–Roothaan self-consistent field (SCF) procedure.

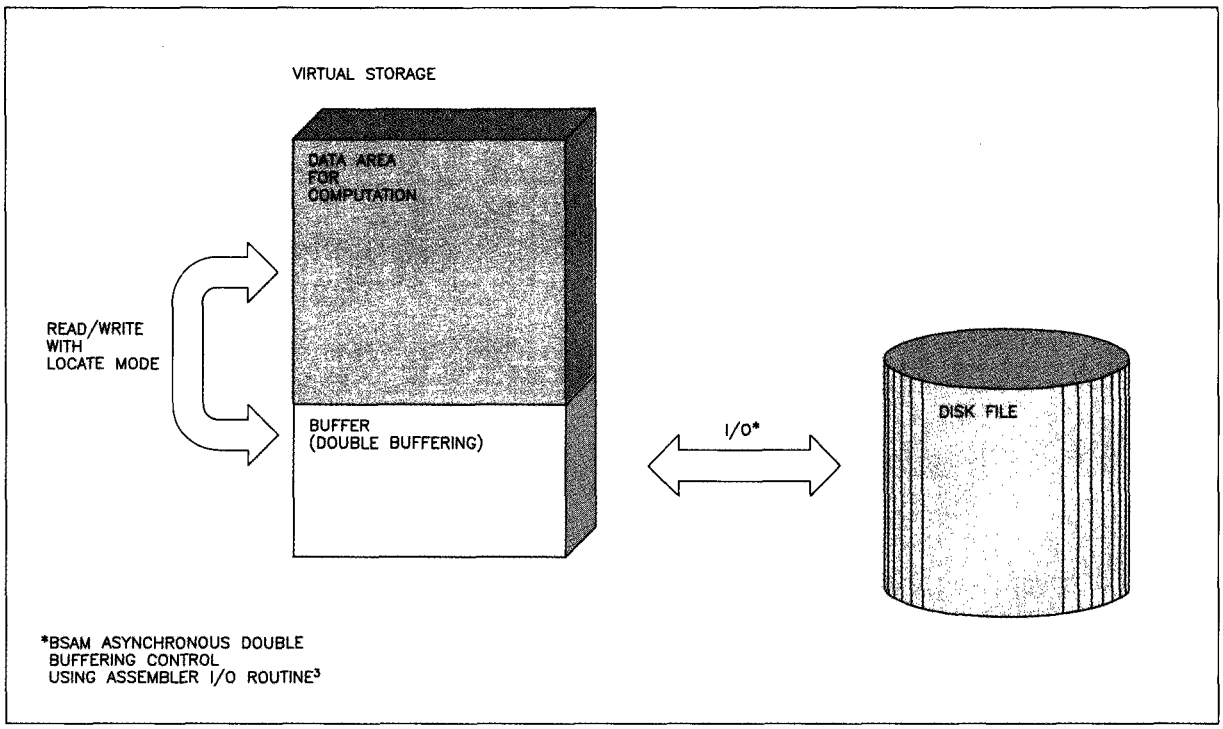
‡ Most of the I/O activities in the MO computations are ascribed to the INT file. Block size of the file is selected as 23 472 bytes.

time distribution analysis revealed that, in cases having more than 60 basis functions, more than 80 percent of the total elapsed time is consumed in the two-electron repulsion integral (TEI) computation step (LINK 311 module) and the SCF iteration step (LINK 501 module). Within these two steps, most of the I/O activity is ascribed to the two-electron INTegral (INT) file. The Elapsed Time/CPU Time ratios are only 1.0–1.2 for the TEI step. However, they come out to be 5.2–12.8 for the SCF step.

At each SCF iteration, all of the INT file records are read. The SCF iteration count ranges from 20–40, depending on the model (see Table 1). Through the analysis, we concluded that the I/O bottleneck was mainly due to the iterative reading of large numbers of TEIs in particular parts of the program (i.e., the SCF step).

Bottleneck analysis. The following I/O technique is used on the TEI (INT file) data access:[3]

Figure 1 Flow of TEI data in the original program



- In the write operation,
  1. The TEI process routine (OUT2E routine) transfers TEI data to one of the toggle buffers, as shown in Figure 1.
  2. When the buffer is filled with data, control is passed to an I/O routine named NTRAN, which transfers integral data to the disk file (INT file). NTRAN is written in assembly language and accesses the INT file sequentially with an asynchronous double-buffering technique.
  3. Control is returned to the OUT2E immediately after NTRAN initiates an I/O operation, and the buffer is switched to continue TEI processing.
- In the read operation, the reverse procedure takes place (i.e., NTRAN transfers integral data from the disk file to the toggle buffers asynchronously).

GAUSSIAN-82 has the capability to optimize the disk I/O speed of the INT file by specifying I/O block size at execution time. The I/O block size is passed to GAUSSIAN-82 through a Job Control Language data control block (JCL DCB) parameter. In general, disk I/O activity is optimized when block size is equal to the full-track or half-track size. According to the rule, we set the block size to 23 472 bytes, which is approximately the half-track size of the IBM 3380 disk. We then observed the I/O operation to the INT file with the block size setting. During the write phase, the TEI computation step had little I/O delay, because the calculation CPU time and I/O block size matched well. During the read phase, the SCF iteration step suffered considerable I/O delay, because the CPU calculation time was too short in comparison to the I/O completion time. The larger the number of basis functions (i.e., the scale of the computations increases), the larger the burden on the SCF iteration step and the longer the total elapsed time became. This was the resulting bottleneck.

New approach to solve the bottleneck. We determined that reduction in disk I/O delay was essential to improve the turnaround time of the numerically intensive computations of *ab initio* MOs. We examined several ways to realize this: disk cache, semiconductor disk, and Expanded Storage. The Expanded Storage looked best because of its speed, capacity, future expandability, and easy modification of source code. Accordingly, we moved the TEI data from the

disk file to a large virtual storage combined with Expanded Storage. In the following section, the modification of the software which led to a noticeable improvement in turnaround time is described.

Design of program modification

New design to exploit Expanded Storage. A large virtual storage combined with Expanded Storage (ES) can handle a vast data area on central and Expanded Storage with V=V addressing architecture in a 31-bit addressing mode (System 370/XA or System 370/ESA[14]). Logically, user data are placed in a large virtual storage area. FORTRAN offers the capability with the *Dynamic Common* (DC) function. Dynamic Common is a large virtual storage dynamically allocated at execution time by FORTRAN. Physically, the logical data are mapped on the MVS/XA storage hierarchy consisting of central storage, Expanded Storage, and disk-paging file. The three storage hierarchy devices are shared among multijob users, and MVS/XA optimizes effective utilization of these three devices based on the LRU (Least Recently Used) algorithm. Initially, MVS/XA tries to keep user data in central storage. When central storage is full, it moves the least inactive data to Expanded Storage page by page (a page size is four kilobytes). If Expanded Storage is large enough to keep the data of all users, almost all of the data will stay on it. But if the total amount of data exceeds the sum of central storage and Expanded Storage, the least inactive data are moved down to the disk-paging file page by page. The following paragraph describes the optimum use of the storage hierarchy based on program characteristics of GAUSSIAN-82.

Dynamic Common allows elapsed time to be reduced to the level of CPU time.

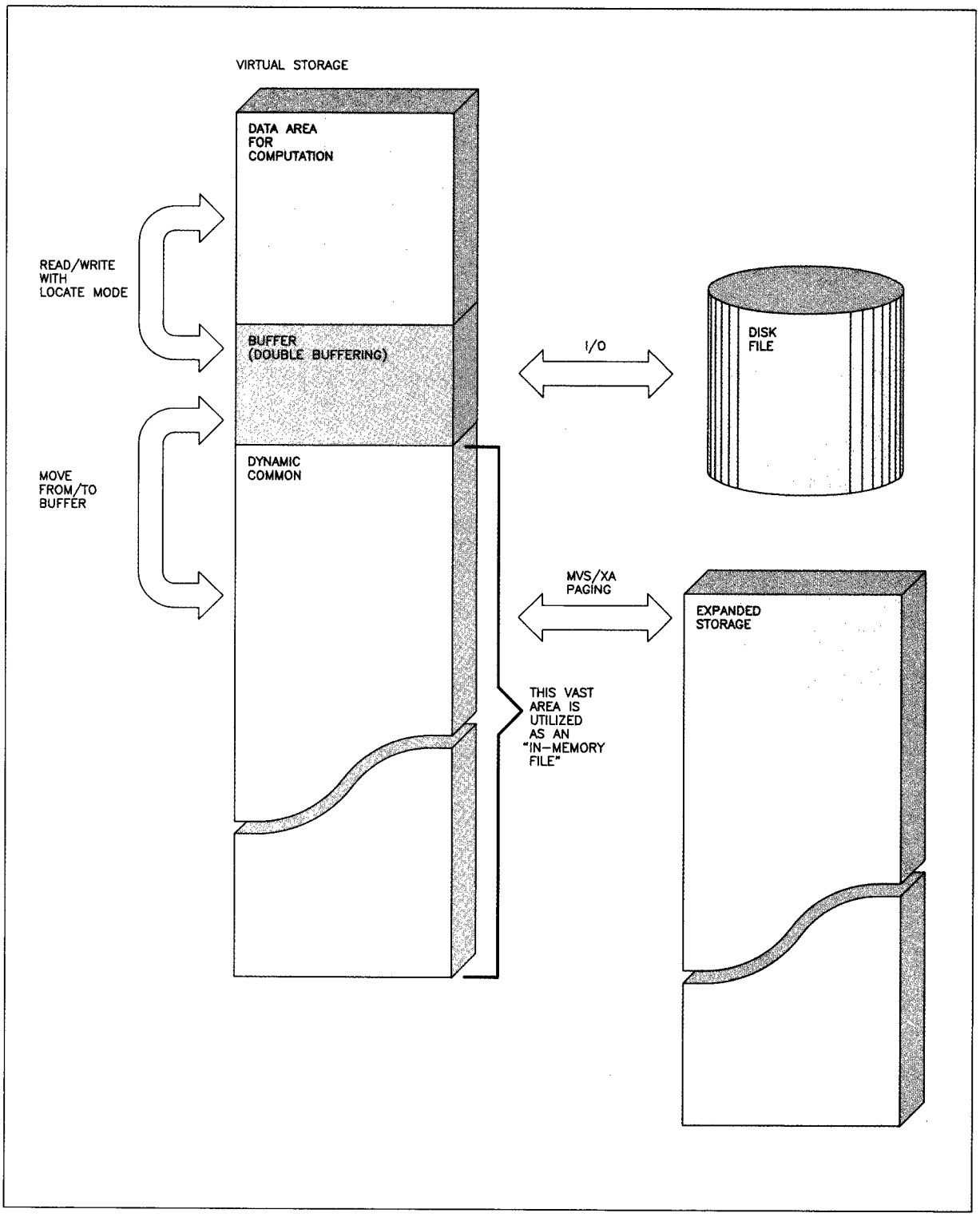
In modifying GAUSSIAN-82, we wanted to optimize data handling for the two-electron repulsion integral (TEI) data so that execution time is improved. The fundamental approach to achieving balanced usage of the CPU, I/O devices, and temporary data set has been discussed, for example, by Shavitt,[15] Yoshimine,[16] McLean,[17] and Bagus et al.[18] But, as described in the previous section, distribution of the TEI data is uniform; i.e., the read/write access pattern is sequential. The data are written once and then read through repeatedly. In addition, the paging capability of MVS/XA was greatly enhanced when Expanded Storage was introduced. On the basis of these points, we selected a simple method that kept the TEI data in Dynamic Common, and let MVS/XA manage it by the paging mechanism.

The next step was to find a way to achieve optimal usage of the MVS/XA paging mechanism. From a system point of view, the paging mechanism is a chained I/O operation of multiple 4-kilobyte (KB) blocks, but it is a sequential I/O operation of 4-KB blocks for an individual program (task). The access speed for Expanded Storage is approximately 75 microseconds per 4-KB page, including MVS/XA routine time,[1] which is 50 to 100 times faster than disk access. Thus, paging to/from Expanded Storage provides a much faster data retrieval capability than any disk I/O operation. In contrast, a disk I/O operation of an INT file (TEI data) is faster than disk paging because the I/O block size of an INT file is large enough (23 KB). The optimal method is to assign the DC size so as to let MVS/XA keep the TEI data either in central storage or Expanded Storage, and not spill the data out to a disk page. In the application we designed, the DC size was determined at execution time through a GAUSSIAN-82 control card, because available DC size may vary according to the customer installation (size of installed expanded storage) and multijob environment. The DC area is considered as an "in-memory file," illustrated in Figure 2. The relationship between elapsed time and CPU time is illustrated in Figure 3. Dynamic Common allows elapsed time to be reduced to the level of CPU time, whereas CPU time remains almost unchanged. In this design, we expected elapsed time to be improved by a factor of three.

Although the size of Expanded Storage on a customer site today may set a ceiling of DC size for effective utilization of the storage hierarchy, we believe it will be overcome by computers in the next generation. Hence, we expect an auspicious future for an approach that will exploit large virtual storage.

Implementation of new design. At the implementation phase, the following two restrictions (the first derived from MVS/XA and the second from GAUSSIAN-82) had to be considered:

Figure 2 Flow of TEI data in the modified program



1. The I/O buffer and I/O routine must be in 24-bit addressing mode and must reside below the 16-MB line.
2. The generalized I/O subroutine named NTRAN was coded in assembler language and looked very difficult to replace with FORTRAN subroutines because of unsupported FORTRAN functions. NTRAN is designed to run with 24-bit addressing mode and resides below the 16-MB line.

On the basis of these restrictions and our design requirements, the following two preconditions were introduced:

1. Dynamic Common (DC) should be placed above the 16-MB line and accessed with 31-bit addressing mode in order to keep a large amount of TEI data in it. It allows the TEI data to increase up to 2 GB.
2. A certain "glue" module was required to make a bridge between the 31-bit addressing DC routines and a 24-bit addressing I/O routine.

We considered several ways to implement the new design satisfying both the restrictions and preconditions.

- Case 1: *FORTRAN DC approach*—Convert FORTRAN programs to the 31-bit addressing mode, allocate the DC area above the 16-MB line, and access it by FORTRAN routines. Add an addressing mode conversion routine so as to make an interface with the NTRAN I/O subroutine.
- Case 2: *Simulating DC with Assembler*—FORTRAN remains with the 24-bit addressing mode. Add Assembler subroutines to switch to the 31-bit addressing mode, and simulate FORTRAN DC with assembler language. The simulated DC is placed above the 16-MB line. Both FORTRAN and Assembler routines reside below the 16-MB line.
- Case 3: *VIO approach*—The program remains unchanged and uses virtual I/O (VIO) for the INT file. (VIO to Expanded Storage was not available when this modification was designed, but we want to include it in our comparison list.)

The three approaches are compared from different viewpoints in Table 3.

We thought splitting the TEI data between Expanded Storage and the disk spill file was essential to achieve high response time. Although VIO requires no modification work, it is not a good candidate for the TEI data because the disk spill file is not available. However, VIO will be very effective for small- or medium-sized calculation of GAUSSIAN-82 where the spill file is not required. Both Cases 1 and 2 looked good for the new approach. We selected Case 2 because of its lighter modification workload and DC size allocation capability (FORTRAN assigns DC size statically at compile time). As a future consideration, if the GAUSSIAN main routine is converted to the 31-bit addressing mode, Case 1 would become very attractive.

The actual program modification was performed as described above.[19] The original code was modified to keep as many integrals as possible in a simulated

Figure 3 Turnaround time improvement through Dynamic Common utilization

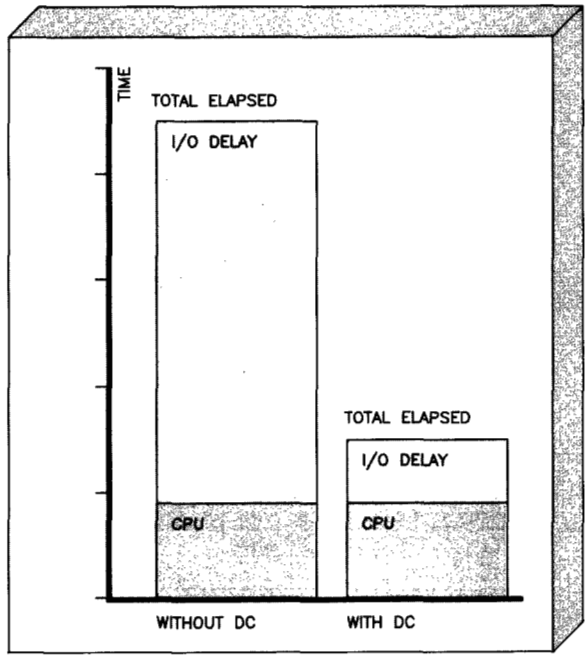


Table 3 Effect of viewpoint on each approach

Viewpoint	Case 1	Case 2	Case 3
Manpower required	large	small	zero
Dynamic Common size	NO	YES	YES
Use of ES	YES	YES	YES
Control of ES data	YES	YES	NO
Disk spill file	YES	YES	NO
Paging unit size	4 KB	4 KB	4 KB
Spilled I/O size	23 KB	23 KB	—

Table 4 Execution analysis of the modified code (unit = second)

Molecule	1-methyl butadiene	phenol	cytosine	guanine
Stoichiometry	C_5H_8	C_6H_6O	$C_4H_5N_3O$	$C_5H_5N_5O$
Used size of Dynamic Common (MB)	12	27	37	99
CPU Time (s)				
TEI step	37.35	74.82	95.47	256.19
SCF step	23.54	55.18	79.41	321.94
Total	66.20	137.88	184.77	598.40
Elapsed Time (s)				
TEI step	39.31	77.31	98.22	260.92
SCF step	35.38	68.42	104.36	586.38
Total	119.82	194.62	252.84	908.61
Original/modified ratio of Total Elapsed Time	1.8	2.2	4.8	5.1
Original/modified ratio of Elapsed Time				
TEI step	1.0	1.0	1.0	1.2
SCF step	3.8	4.6	10.2	7.4
Ratio of Total Elapsed Time to Total CPU time	1.8	1.4	1.4	1.5
Ratio of Elapsed Time to CPU Time				
TEI step	1.1	1.0	1.0	1.0
SCF step	1.5	1.2	1.3	1.8
Weight percent of CPU Time				
TEI step	56	54	52	43
SCF step	36	40	43	54
Sum	92	94	95	97
Weight percent of Elapsed Time				
TEI step	33	40	39	29
SCF step	30	35	41	65
Sum	63	75	80	94

DC, and remaining integrals (spilled-over integrals) were transferred to/from an IBM 3380 disk file. With the implementation, we developed four Assembler routines and seven FORTRAN routines, and added a small common storage to keep control information used for DC simulation. The following four Assembler routines were developed to cover DC simulation:

1. DCOPEN—Establish 31-bit addressing mode and create a DC area.
2. DCWRIT—Establish 31-bit addressing mode and move a record from the FORTRAN buffer to DC.
3. DCREAD—Establish 31-bit addressing mode and move a record from DC to the FORTRAN buffer.
4. DCLOSE—Establish 31-bit addressing mode and release the DC area.

The following three FORTRAN routines are modified for job control:

1. DOLLAR—Read DC control cards to determine DC size.
2. CHAINX—Write common data to a disk file in order to pass common data to a user-controlled overlay module.
3. DRUM—Read common data from a disk file in order to receive common data from a user-controlled overlay module.

The following four FORTRAN routines are modified to perform access control to DC and the disk spill file:

1. IWRIT—Control the write operation by splitting the TEI data to DC or to the disk spill file.
2. IREAD—Control read operation by merging the TEI data from DC and the disk spill file.
3. IWIND—Reset DC pointer and rewind the disk spill file.
4. IWAIT—Synchronize I/O completion. No synchronization is necessary for DC area access.

Effects of modification

Measurement of the modified code. Execution analysis has been performed with the modified code. The sample molecules and the method of computation are the same as in the analysis of the original code, which was described earlier. The resultant timing data are given in Table 4. By taking guanine in the table as an example, the relationship between total elapsed time and total CPU time before and after Dynamic Common utilization is illustrated in Figure 4, and that between elapsed time and CPU time at the SCF step in Figure 5. CPU time improvement at the SCF step comes partly from I/O activity reduction and partly from VF utilization.[20]

Noticeable improvement in turnaround time. We can point to the following evidence of turnaround time

Figure 4 Total turnaround time improvement through Dynamic Common utilization in the case of guanine

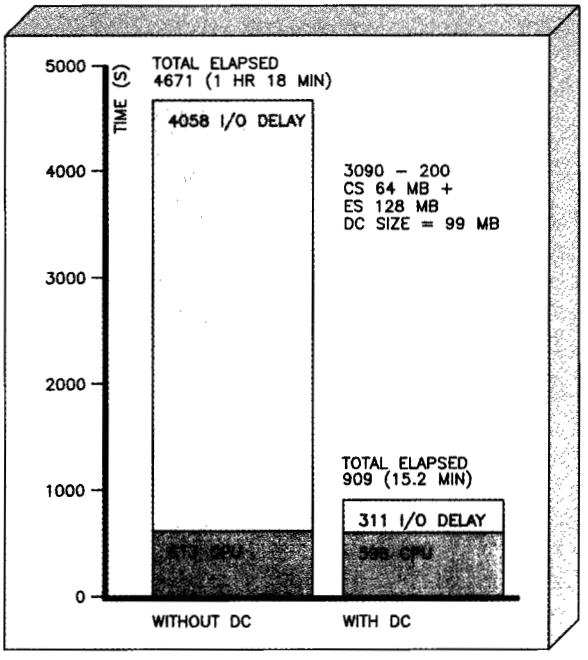
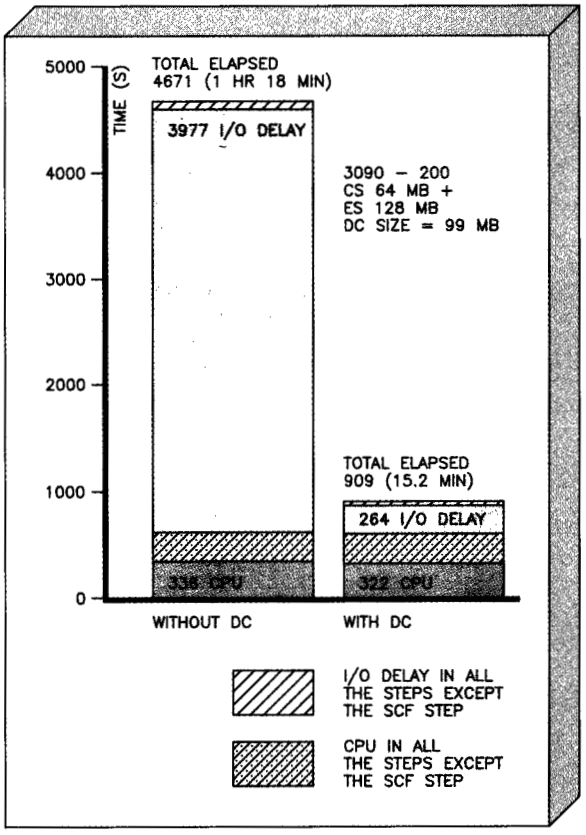


Figure 5 Turnaround time improvement through Dynamic Common utilization at the SCF step in the case of guanine

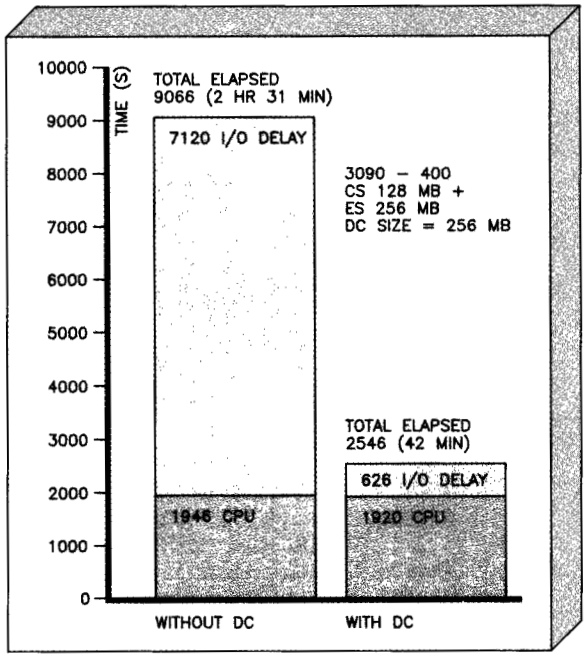


improvement through this analysis:

1. The ratio of total elapsed time with the original code to that with the modified code (Original/Modified ratio) shows that the total elapsed time with the modified code becomes 1.8–5.1 times shorter than that with the original code. This reveals a substantial improvement in total turnaround time.
2. The Elapsed Time/CPU Time ratio of the SCF step is reduced from 5.2–12.8 to less than 2.0. The Total Elapsed Time/Total CPU Time ratio is also reduced from 3.0–7.6 to less than 2.0. These numbers indicate that the present modification has almost removed the I/O bottleneck found in the molecular orbital computations.

We find from Table 4 that the CPU time is primarily consumed (over 90 percent) in the TEI and SCF step.

Figure 6 **Total turnaround time improvement through Dynamic Common utilization in the case of phenol with large basis set**



We also find that elapsed time in the SCF step with the modified code has achieved a 3.8- to 10.2-times improvement against the original code. This indicates that the first result is mainly due to the improvement in the turnaround time at the SCF step, since the elapsed time in the TEI step is close to the CPU time even with the original code.

It is observed from the computations at the SCF step (using the LINK 501 module) that the CPU time per one SCF iteration increases roughly by the 3.4 power of the number of basis functions. The Elapsed Time/CPU Time ratios of the original code are observed to be over five at the SCF step. They are expected to remain over five even in cases of much-larger-scale computations, in which a greater number of basis functions is employed, for example, to obtain higher accuracy or to apply to larger-sized molecular systems. Dynamic Common utilization for the SCF step would be more effective for such larger-scale computations, because much more turnaround time should be saved by the utilization.

In this regard, a larger-scale computation has been performed for the phenol molecule using the 6-31G** basis set[4] (135 basis functions and 238 primitive Gaussian functions). The TEI step generated 20 631 816 TEIs, and the SCF step required 24 SCF iterations for the convergence. The size of the Dynamic Common utilization necessary for storing all of the TEIs was 248 megabytes. This computation was carried out on a 3090-400 with two Vector Facilities. The resultant improvement in the timing is illustrated in Figure 6. The results showed that the elapsed time with the DC (2525 seconds) was 3.6 times shorter than without the DC (9066 seconds). Thus, the effectiveness of the Dynamic Common utilization is confirmed in the numerically intensive computations of *ab initio* molecular orbitals.

Summary and concluding remarks

The IBM 3090 introduced a new technology, called Expanded Storage, with a capacity up to 2 GB, in the group of computers classified as large processors. It is a new level of storage hierarchy, and the MVS/XA operating system utilizes it as high-speed paging equipment, allowing a user to hold application data in large virtual storage (FORTRAN Dynamic Common area). In order to exploit the large virtual storage capability of the IBM 3090, we intended to develop *a new application technique* for numerically intensive computations of *ab initio* molecular orbitals. We modified an application program (GAUSSIAN-82) on the 3090 MVS/XA so that it can handle a vast amount of intermediate data in large virtual storage combined with Expanded Storage; i.e., application data were stored in a large virtual storage instead of disk file storage. The modification achieved a 4- to 10-fold improvement in turnaround time at a CPU rate-determining step (SCF step) in medium-sized molecules. Thus, the advantage of large virtual storage combined with Expanded Storage was demonstrated. The present modification will be quite effective for the computation of larger molecular systems. With the use of large virtual storage, a similarly noticeable improvement in turnaround time can be expected for other applications where the computations have an I/O bottleneck due to excessive disk access.

Acknowledgments

The authors would like to thank Vector Facility task members in IBM Japan for discussions at the early development stage, members in SDSC who prepared for the MVS/XA system, members in STC who managed machine usage time, A. Koide (Tokyo Scientific Center) for providing a program modification of the ESSL call, and R. Hosoi (Tokyo NIC Center) for his kind help in the preparation of this paper.

MVS/XA is a trademark of International Business Machines Corporation.

Cited references and notes

1. S. G. Tucker, "The IBM 3090 system: An overview," *IBM Systems Journal* **25,** No. 1, 4–19 (1986).
2. D. H. Gibson, D. W. Rain, and H. F. Walsh, "Engineering and scientific processing on the IBM 3090," *IBM Systems Journal* **25,** No. 1, 36–50 (1986).
3. J. S. Binkley, M. Frisch, K. Raghavachari, D. DeFrees, H. B. Schlegel, R. Whiteside, E. Fluder, R. Seeger, and J. A. Pople, "GAUSSIAN 82. Release A," Carnegie Mellon University, Pittsburgh (1983).
4. W. J. Hehre, L. Radom, P. v. R. Schleyer, and J. A. Pople, *Ab Initio Molecular Orbital Theory*, John Wiley & Sons, Inc., New York (1986).
5. C. C. J. Roothaan, "New developments in molecular orbital theory," *Review of Modern Physics* **23,** No. 2, 69–89 (1951).
6. E. Clementi, "Computer simulations of complex chemical systems: Solvation of DNA and solvent effects in conformational transitions," *IBM Journal of Research and Development* **25,** No. 4, 315–326 (1981), and references given therein.
7. P. S. Bagus and A. R. Williams, "Electronic structure theory," *IBM Journal of Research and Development* **25,** No. 5, 793–809 (1981), and references given therein.
8. P. S. Bagus, B. Liu, A. D. McLean, and M. Yoshimine, in *Computational Methods for Large Molecules and Localized States in Solids*, E. Herman, A. D. McLean, and R. K. Nesbet, Editors, Plenum Press, New York (1973).
9. *Computational Methods in Chemistry*, J. Bargon, Editor, IBM Research Symposia Series, Plenum Publishing Company, New York (1980).
10. The software programs which have been developed at IBM and have been widely used on IBM computers are exemplified by those written by Clementi et al. (10a–10g), by McLean and Yoshimine (10h), by Nesbet (10i), by Bagus, Liu, McLean, and Yoshimine (10j), by Liu and Yoshimine (10k), and so on: (a) E. Clementi and C. Roetti, *IBM Journal of Research and Development* **9,** 2 (1965) and supplement; (b) E. Clementi and D. R. Davis, *Journal of Computational Physics* **1,** 223 (1966); (c) E. Clementi, *Journal of Chemical Physics* **46,** 3851 (1967); (d) J. W. Mehl and E. Clementi, *IBM System/360 IBMOL-5 Program Quantum Mechanical Concepts and Algorithms*, IBM Technical Report RJ-883 (1971); (e) R. Pavani and L. Gianolio, *IBMOL-6 Program—User's Guide*, 1st edition, Ricerche G. Donegani, Novara, Italy, Technical Report DDC-771 (1977); (f) E. Clementi, G. Corongiu, J. Detrich, S. Chin, and L. Domingo, *International Journal of the Quantum Chemistry Symposium* **18,** 601 (1984); (g) *KGNMOL*; R. Gomperts and E. Clementi, *Quantum Chemistry Program Exchange (QCPE)* **8,** 33 (1988); (h) *McL-YOSH LINEAR MOLECULE PROGRAM 1*, A. D. McLean and M. Yoshimine, *IBM Journal of Research and Development* **12,** 206–233 (1968); (i) R. K. Nesbet, in *Advances in Quantum Chemistry*, P. O. Löwdin, Editor, Volume 3, Academic Press, Inc., New York (1967), and references given therein; (j) *ALCHEMY*, P. S. Bagus, B. Liu, A. D. McLean, and M. Yoshimine, see Reference 9; (k) *ALCHEMY II*, B. Liu and M. Yoshimine, *Journal of Chemical Physics* **74,** 612 (1981); B. H. Lengsfield and B. Liu, *Journal of Chemical Physics* **75**, 478 (1981).
11. There are *ab initio* MO programs which have been developed outside IBM and widely used on IBM computers, including GAMESS (11a), HONDO (11b, c), GAUSSIAN (3, 4), and so on (11d): (a) M. Dupuis, D. Spangler, and J. J. Wendoloski, *National Resource for Computations in Chemistry, Software Catalog*, Volume 1, program QG01 (1980); (b) *HONDO5*, M. Dupuis, J. Rys, H. F. King, *QCPE* **13,** 403 (1981); (c) *HONDO (Version 7.0)*, M. Dupuis, J. D. Watts, H. O. Villar, and G. J. B. Hurst, *QCPE* **8,** 79 (1988); HONDO (Version 7.0) was developed by M. Dupuis, currently at IBM in Kingston, NY; (d) for example, see *QCPE Catalog*, Department of Chemistry, Indiana University, Bloomington, IN 47405.
12. *IBM VS FORTRAN Execution Analyzer, Program Description/Operations Manual*, SC23-0335 (Program Number 5798-DXJ), IBM Corporation (March 1986); available through IBM branch offices.
13. The condition under which the analysis was performed is as follows: (1) The block size of the TEI file (INT) was selected as 23 472 bytes. (2) Default input data which were provided by GAUSSIAN-82 were used except for the quantum mechanical method specification (RHF), basis set specification (3-21G), and molecular geometry specification. Molecular geometries were specified by the use of the model builder (MODEL-A) of the GAUSSIAN-82.[3] The point group of *Cs* was enforced to the electronic wavefunctions. The Raffenetti format was chosen for the INT file format, since the SCF processing time could be reduced.[3]
14. *MVS/ESA General Information for SP Version 3*, GC28-1359, IBM Corporation (1988); available through IBM branch offices.
15. I. Shavitt, Chapter 6 in *Modern Theoretical Chemistry, Vol. III, Methods of Electronic Structure Theory*, H. F. Schaefer, Editor, Plenum Press, New York (1977), and references given therein.
16. M. Yoshimine, "Construction of the Hamiltonian Matrix in large configuration interaction calculations," *Journal of Computational Physics* **11,** No. 3, 449–454 (1973).
17. A. D. McLean, in *Proceedings of the Conference on Potential Energy Surface in Chemistry*, University of California, Santa Cruz, CA (August 1970); *Technical Publication RA18*, IBM Research Division Laboratory, San Jose, CA (1971).
18. P. S. Bagus, B. Liu, A. D. McLean, and M. Yoshimine, in *Energy, Structure, and Reactivity*, D. W. Smith and W. B. McRae, Editors, John Wiley & Sons, Inc., New York (1973), p. 130.
19. M. Sakaki, *GAUSSIAN-82 3090-VF Modification Design Manual*, Data Systems Support Center, IBM Japan, Ltd., Tokyo (1986).
20. The Vector Facility (VF) was used only for a matrix eigenvalue/eigenvector calculation at a computational step. ESSL (Engineering and Scientific Subroutine Library) routines were called in the modified program, and a reasonable effect of the VF was observed. Whereas this implies a possible increase of CPU speedup with VF utilization, the dominant contribution to the improvement in turnaround time is attributed to the expanded storage. In this paper, the focus is on the application of the expanded storage.

Mikio Sakaki *IBM Japan Limited, Systems Engineering, Kowa Tsukiji Building, 7-18-24, Tsukiji, Chuoh-ku, Tokyo 104, Japan.*

Mr. Sakaki is currently a senior advisory systems engineer in the Systems Design Support Center. Since he joined IBM Japan in 1973, he has participated in marketing and installation support of large central processors (M168, 3033, 308X, 3090) and operating systems (VS1, SVS, MVS, MVS/XA). He is currently working on the analyses of various processors and on marketing support of the Vector Facility (VF) and IBM 3090. He received his B.E. degree in mathematical engineering from Kyoto University.

Hikaru Samukawa *IBM Japan Limited, NIC Marketing Center, Shinjuku Sumitomo Building, 6-1, Nishi-Shinjuku 2-chome, Shinjuku-ku, Tokyo 163, Japan.* Mr. Samukawa is an advisory industry specialist in the NIC Marketing Center, where he is working on marketing activities for the 3090 Vector Facility. He joined IBM Japan in 1984 and has been working on the Vector Facility since 1985. Mr. Samukawa worked as a programmer/designer of numerical analysis software such as NASTRAN, etc. at UNIVAC Japan. He received his B.S. in mechanical engineering from Waseda University in 1972. Mr. Samukawa is a member of the Japan Information Processing Society.

Nobumitsu Honjou *IBM Japan Limited, Tokyo Research Laboratory, 5-19, Sanban-cho, Chiyoda-ku, Tokyo 102, Japan.* Dr. Honjou is currently a researcher at the Tokyo Scientific Center. After he joined IBM Japan in 1984, he worked on projects such as IBM Japan's participation in the Tsukuba Scientific EXPO'85, and scientific applications. Prior to joining IBM Japan, he worked in the quantum chemistry group at the IBM San Jose Research Laboratory as an IBM World Trade Postdoctoral Fellow from 1981 to 1982. Dr. Honjou received his B.S. degree in chemistry from Shizuoka University, and his M.S. and D.S. degrees in chemistry from Hokkaido University.

Reprint Order No. G321-5341.