Theor Chim Acta (1987) 71: 91–103

Theoretica Chimica Acta

Parallel computation of molecular energy gradients on the loosely coupled array of processors (LCAP)

M. Dupuis and J. D. Watts

IBM Corporation, Data Systems Division, Scientific and Engineering Computations, Department 48B, Mail Stop 428, Kingston, NY 12401, USA

(Received May 7, revised August 27/Accepted September 29, 1986)

The implementation of the HONDO program on the Loosely Coupled Array of Processors (LCAP) parallel computer system assembled in our laboratory is presented. We discuss a general strategy used to maintain a high level of compatibility between the serial version and the parallel version of the code. We report the implementation of energy gradient calculation for SCF wavefunctions. The integral and integral derivative programs display high parallel efficiency, and so does the SCF part in the case of very large basis sets.

Key words: Energy gradient calculation — SCF wavefunction — Parallel computer system

1. Introduction

For some years now SCF energies and energy gradients of small and medium-sized molecules have been routinely calculated in many laboratories. The availability of efficient, reliable, "black-box" computer programs [1] and increasing computer power have enabled increasingly complex molecular systems to be studied computationally. At the same time, the molecular scientist's expectations have increased, and the demand for computing facilities is greater than ever. Fortunately, the CPU intensive parts of many scientific applications programs in general, and SCF energy and energy gradient programs in particular, are well suited to vector and parallel computer architectures.

Recently, two powerful parallel computer systems based on a loosely coupled array of processors (LCAP) have been assembled in this laboratory [2–4]. The first of these systems, LCAP1, presently comprises an IBM 3081 front-end computer and ten FPS-164 attached processors and runs under the VM operating

system. The second system, LCAP2, consists of an IBM 3084 and ten FPS-264 attached processors and runs under the MVS operating system. Each system supports two modes of parallel execution. Most commonly, an IBM processor coordinates execution of a single job on two or more of the attached processors. Inboard parallelism, whereby a single job makes use of the two processors of the 3081 or the four processors of the 3084, is also available. Even at this early stage the general applicability of LCAP has been amply demonstrated [4–10]. Many scientific and engineering applications programs have been migrated from sequential to parallel, and a wide range of computationally demanding problems have been/are being tackled with the aid of LCAP. The applicability of the system is expected to increase still further with the addition of large shared memories and other facilities which assist interprocessor communication [4].

The first major success of the LCAP experiment was the migration from sequential to parallel of the integral and SCF modules of the quantum chemical program IBMOL [2, 3, 5]. Through this work it was possible, for example, to make a detailed *ab initio* study of proton tunneling in DNA base pairs [4–7] an investigation for which supercomputer performance was imperative.

In this report we describe the parallelization of the integral, SCF and integral derivative modules of the HONDO program [11]. This work extends that of Clementi et al. [4,5] by providing parallel computation of not only the SCF energy and wave function, but also the energy gradient. The importance and value of the energy gradient have been appreciated for some time [12]. The gradient is necessary for efficient location of stationary points on potential energy surfaces, for the determination of reaction pathways, and it can be used to obtain force constants and vibrational frequencies by a finite differencing procedure. If *sp* basis sets are used, the times for SCF energy and gradient evaluation are comparable [13]. When polarization functions are included in the basis set the computation of the gradient usually takes substantially longer than energy evaluation [13].

The strategy used in the parallelization of the IBMOL integral program evolved very naturally from the loops over nuclear centres which drive the sequential code. Several other integral computation codes, for example those in the HONDO [11] and the GAUSSIAN 82 programs [14], are based on the so-called shell structure. A shell comprises all the basis functions located on a given centre having the same contraction coefficients and exponents. Integral programs using the shell structure are also easily adapted for parallel execution by distributing the loop over shell blocks among the processing units. Compared with the "centre approach", the "shell approach" leads to parallelism with a finer but more uneven grain. This is because high angular momentum integrals are more CPU demanding than low angular momentum integrals and high angular momentum shells lead to many more integrals than low angular momentum shells. It is possible, therefore, that parallelization of an integral program based on the shell approach may produce a well balanced load as far as the CPU time is concerned, even though the numbers of integrals evaluated on the different processors may not be equal.

Clearly, a gross imbalance in the numbers of integrals on the different processors will lead to poor load balancing in the SCF program.

The most time consuming step in the calculation of the derivatives of the SCF energy with respect to the nuclear coordinates is the evaluation of the derivatives of the two-electron integrals with respect to nuclear coordinates. Here too the shell structure dictates the computational strategy in a manner very similar to the integral program. Each integral derivative is evaluated and multiplied by an appropriate density factor, and the result is added to the energy gradient vector. In addition to the basis function parameters, the derivative programs need only the density matrix. It is clear, then, that the gradient program should also be easily parallelized and that the data transfer overhead is minimal.

The plan of this report is as follows. Section 2 contains a brief summary of the working equations of the program. Section 3 describes a programming technique we used for the parallel implementation of HONDO, a program with over 75 000 lines of FORTRAN code. This technique made it very easy to parallelize the code and to maintain exact compatability between serial and parallel versions. This programming technique should prove extremely useful and convenient for handling large FORTRAN codes. In Sect. 4 we present and discuss benchmark performances for calculations on a range of molecular systems.

2. Equations for the evaluation of the SCF energy and energy gradient

Detailed discussion and derivation of the equations used for the calculation of SCF energies and energy gradients can be found elsewhere [12, 15–19]. For future reference and to define more clearly our computational task we give a brief summary here. We restrict our attention to SCF wave functions for closed-shell systems. The theory for more general SCF wave functions is not substantially more complex and excellent discussions are given elsewhere [18, 19]. Throughout this section μ, ν, λ, and σ are used to represent atomic basis functions and i and j to label molecular orbitals.

For closed-shell molecular systems the SCF energy is given by

$$E = \sum_{\mu\nu} D_{\mu\nu}(\mu/h/\nu) + \frac{1}{2} \sum_{\mu\nu\rho\sigma} D_{\mu\nu} D_{\rho\sigma}[2(\mu\nu/\rho\sigma) - (\mu\rho/\nu\sigma)] + V_{NUC} \tag{1}$$

In this expression V_{NUC} is the nuclear repulsion energy, the density matrix elements $D_{\mu\nu}$ are defined in terms of the orbital coefficients by

$$D_{\mu\nu} = \sum_{i}^{occ} C_{\mu i} C_{\nu i} \tag{2}$$

and the $(\mu/h/\nu)$ and $(\mu\nu/\rho\sigma)$ are respectively the one- and two-electron integrals. The molecular orbital coefficients are defined iteratively by diagonalizing the Fock matrix $\boldsymbol{F}$, the elements of which are given in the basis functions representation by

$$F_{\mu\nu} = (\mu/h/\nu) + \sum_{\mu\nu\rho\sigma} D_{\rho\sigma}[2(\mu\nu/\rho\sigma) - (\mu\rho/\nu\sigma)]. \tag{3}$$

The computationally demanding steps in the calculation of the SCF energy are the evaluation of the two-electron integrals, typically there are several million of these, and the contraction of these integrals with density matrix elements to form the Fock matrix. The diagonalization of the Fock matrix may also require a significant computation time. Once the SCF procedure has converged the derivatives of Eq. (1) with respect to nuclear coordinates can be calculated from

$$\frac{\partial E}{\partial q}=\sum_{\mu\nu} D_{\mu\nu}\frac{\partial}{\partial q}(\mu/h/\nu)+\frac{1}{2}\sum_{\mu\nu\rho\sigma} D_{\mu\nu}D_{\rho\sigma}\frac{\partial}{\partial q}[2(\mu\nu/\rho\sigma)-(\mu\rho/\nu\sigma)]$$

$$+\sum_{\mu\nu} W_{\mu\nu}\frac{\partial}{\partial q}S_{\mu\nu}. \tag{4}$$

The elements of the energy-weighted density matrix (or Lagrangian) $\boldsymbol{W}$ are given by

$$W_{\mu\nu}=\sum_{i}^{occ}\varepsilon_i C_{\mu i}C_{\nu i} \tag{5}$$

and $S_{\mu\nu}$ are the overlap matrix elements. The orbital energies ε_i are the eigenvalues of the Fock operator matrix $\boldsymbol{F}$. By far the most time consuming step in the evaluation of Eq. (4) is the computation of the derivatives of the two-electron integrals.

3. Parallelization

We start this section by describing our general approach to parallelization. Following this we make a few remarks on how we have migrated the individual modules from sequential to parallel.

So far we have prepared two parallel versions of the program, one for LCAP1 and one for LCAP2. In both versions of the program the processing units are the FPS-164s (LCAP1) or FPS-264s (LCAP2). Versions of the program which execute in parallel on the different processing units of the multiprocessor 3081 and 3084 computers are in preparation. Although we talk of different versions of the program, there is in fact only one version of the code; there are differences in how one runs the program under the two operating systems. In the following the IBM front-end computers will be referred to as the host or master and the FPS machines as APs (attached processors or attached array processors). In our work we have benefitted from the use of the precompiler recently developed in this laboratory [20]. The precompiler greatly simplifies the development of parallel programs.

A. General strategy

Like many other applications programs, HONDO begins with some initialization followed by calls to several CPU intensive subroutines (e.g. the integral, SCF, and integral derivative programs). These subroutines are, of course, the parts of the code one wishes to run in parallel. Obviously, one may attempt to parallelize

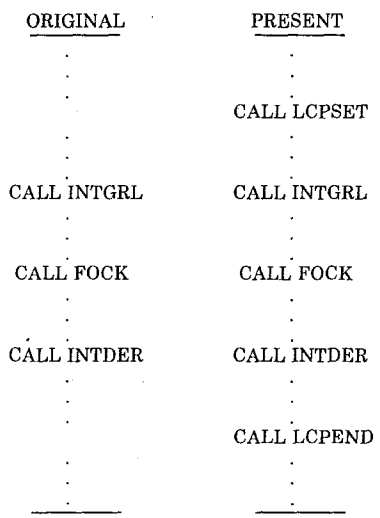

Fig. 1. Original and present serial code (schematic)

the code by inserting the precompiler directives and other code needed for parallel execution in the sequential code. While this approach can undoubtedly be made to work, we believe that another approach is more desirable, particularly for large programs such as HONDO.

Our first step was to make some modifications to the sequential program, but these were very minor changes and do not in any significant way affect how the program runs; the modified sequential program runs alone on the host machine exactly as the original sequential program and is shown in Fig. 1-3. The changes made to the sequential code were the addition of two short subroutines called LCPSET and LCPEND, the addition of a common block called LCAPID into subroutines INTGRL, FOCK, and INTDER, and the insertion of a conditional if statement in the loops over shell blocks in INTGRL and INTDER. Since the new sequential program has all the facilities of the original sequential program, the latter has been discarded, and from now on whenever we refer to the sequential program we imply this slightly modified version.

The next stage in the parallelization was to write new subroutines called LCPSET, LCPEND, INTGRL, FOCK, and INTDER. The new versions of LCPSET and LCPEND respectively initialize and terminate communication between master

```
      SUBROUTINE LCPSET
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/LCAPID/IAP,NAP
      IAP=1
      NAP=1
      RETURN
      END

      SUBROUTINE LCPEND
      IMPLICIT REAL*8 (A-H,O-Z)
      RETURN
      END
```

Fig. 2. Subroutines LCPSET and LCPEND as in present serial code

```
   SUBROUTINE INTXXX                SUBROUTINE INTXXX
                                    COMMON/LCAPID/IAP,NAP
      .                                .
      .                                .
      .                                .
      .                                .
      .                                .
   IJKLSH=0                         IJKLSH=0
   DO 9000 II=1,NSHELL              DO 9000 II=1,NSHELL
      .                                .
      .                                .
      .                                .
   DO 8000 JJ=1,II                  DO 8000 JJ=1,II
      .                                .
      .                                .
      .                                .
   DO 7000 KK=1,II                  dO 7000 KK=1,II
      .                                .
      .                                .
      .                                .
   MAXLL=KK                         MAXLL=KK
   IF(KK.EQ.II) MAXLL=JJ            IF(KK.EQ.II) MAXLL=JJ
   DO 6000 LL=1,MAXLL               DO 6000 LL=1,MAXLL
      .                                .
      .                                .
      .                                .
   IJKLSH=IJKLSH+1                  IJKLSH=IJKLSH+1
                                    IF(MOD(IJKLSH,NAP)+1.NE.IAP) GO TO 6000
      .                                .
      .                                .
      .                                .
6000 CONTINUE                    6000 CONTINUE
7000 CONTINUE                    7000 CONTINUE
8000 CONTINUE                    8000 CONTINUE
9000 CONTINUE                    9000 CONTINUE
      .                                .
      .                                .
      .                                .
   RETURN                           RETURN
   END                              END
```

Fig. 3. The loop over shells in the original and present serial codes (subroutines INTGRL and INTDER)

and slaves. The new versions of INTGRL, FOCK, and INTDER are short "fake" subroutines which contain precompiler directives and a few other lines of code needed to enable subroutines INTGRL, FOCK, and INTDER to be run in parallel. Among other things, the precompiler directives are translated into calls to APROUTINEs. The APROUTINEs also consist only of a few lines of code. They execute on the APs, define the data to be transferred between master and APs, and call AP Fortran versions of subroutines INTGRL, FOCK, and INTDER. In summary, the parallel program comprises

(i) the entire sequential code
(ii) the new subroutines LCPSET, LCPEND, INTGRL, FOCK, and INTDER
(iii) the APROUTINEs APINT, APFCK, and APDER
(iv) AP Fortran versions of subroutines INTGRL, FOCK, and INTDER

and is shown in Fig. 4. It should be mentioned that INTGRL, FOCK, and INTDER are not the names of actual subroutines. Rather, they are single names denoting one of several possible driver routines for integral evaluation (JKINT or PKINT), Fock matrix formation (HSTAR, HSTARU, HSTARO), or integral

	MASTER	ATTACHED PROCESSOR (FPS-164 or -264)
SERIAL CODE	PRECOMPILER INPUT FILE HNDM FORTRAN	PRECOMPILER INPUT FILE HNDA FORTRAN
.		
.		
CALL LCPSET	SUBROUTINE LCPSET	
.	C initialize /LCAP$M/	
.	C Begin parallel	
.	C processing	
.	C$ START	
.	RETURN	
.	END	
.		
.		
CALL INTGRL	SUBROUTINE INTGRL	APROUTINE APINT
.	C$ EXECUTE ON ALL,	CALL INTGRL
.	C$ 1 USING IAP:APINT	RETURN
.	RETURN	END
.	END	
.		
.		
CALL FOCK	SUBROUTINE FOCK	APROUTINE APFCK(F)
.	C$ EXECUTE ON ALL,	CALL FOCK
.	C$ 1 USING IAP:APFCK(F)	RETURN
.	C$ ADDING F	END
.	RETURN	
.	END	
.		
.		
CALL INTDER	SUBROUTINE INTDER	APROUTINE APDER(DE)
.	C$ EXECUTE ON ALL,	CALL INTDER
.	C$ 1 USING IAP:APDER(DE)	RETUR
.	C$ ADDING DE	END
.	RETURN	
.	END	
.		
.		
CALL LCPEND	SUBROUTINE LCPEND	
.	C Terminate parallel	
.	C processing	
.	C$ FINISH	
.	RETURN	
.	END	

Fig. 4. Structure of the parallel program

derivative computation (JKDER). In the same way, the names of the APROUTINEs (APINT, APFCK, and APDER) are purely symbolic.

The success of our scheme depends on the fact that it is possible to load more than one subroutine with the same name; if two or more versions of a subroutine are loaded all but the first version are ignored; that is, to run the parallel program we load the routines listed in (ii) before the sequential object code.

Our strategy has several attractive features, e.g., simplicity, ease of debugging during parallelization, the fact that only a very small portion of the code need be submitted for precompilation (the routines listed in (ii) and (iii)), and compatability with the sequential program (albeit a sequential program primed for parallelization!)

B. The two-electron integral program

The HONDO program employs two methods of two-electron integral evaluation, the rotating axis method of Pople and Hehre [21] first implemented in the GAUSSIAN 70 program [22] and restricted to basis sets containing *s* and *p* functions, and the more general method of Rys quadrature developed by Dupuis, Rys, and King [23-27].

In both the Rys and GAUSSIAN 70 schemes the integrals are evaluated most efficiently by dividing the basis functions into shells and looping over unique shell blocks. The integrals of one shell block may be evaluated independently from those of another shell block and so the integral programs were easily parallelized by distributing the loop over shell blocks among the different APs. This is accomplished in the following way. Suppose that there are NAP APs labelled by IAP and that the unique shell blocks are labelled by IJKLSH. Each AP has a loop over all shell blocks (Fig. 3) but only evaluates the IJKLSH*th* shell block if

$$\text{MOD(IJKLSH, NAP)} + 1 = \text{IAP}. \tag{4}$$

Evidently, the same approach could be achieved by having the loop on the host, but this would necessitate much more host-slave communication with consequent loss of efficiency. The fact that all the APs have the entire four-fold loop over shells introduces a small overhead which cannot be seen in the timing data. The only difference between the AP and sequential code is the change needed to pack the integral labels into 64 rather than 32 bit words. This change has nothing to do with the parallelization and would be necessary if one were writing an FPS-164 version of the sequential code; it is necessary solely because of the different integer word lengths of FPS and IBM machines.

C. The SCF program

There are two computationally demanding tasks in the SCF procedure, the manipulation of the two-electron integrals to form the Fock matrices and the diagonalization of these matrices. At the present stage in our work only the former process is executed in parallel. Intuitively this is reasonable since the Fock matrix formation is an n^4 process while the diagonalization time increases only as n^3 (where n is the number of basis functions). However, like Clementi et al. [4, 5], we have found that the diagonalization and other sequential code may account for a significant fraction of the total execution time, with consequent decrease in efficiency as the number of APs is increased (see Sect. 4).

To parallelize the Fock matrix formation we used the same straightforward strategy as used in the parallelization of this part of the IBMOL program [4, 5]. That is, each of the APs evaluates the contribution of its own two-electron integral list to the total Fock matrix. The latter is obtained by adding together the NAP contributions. Apart from the modifications needed to unpack the integral labels from a 64 rather than a 32 bit word, the AP Fock matrix formation code is exactly the same as in the sequential program.

Table 1. Execution times, speedup factors, and efficiencies for the integral, SCF, and integral derivative programs in the calculations on CROWN compound (LCAP1, 3081 host; LCAP2, 3084 host)[a]

	Integral			SCF			Integral derivative		
NAP	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.
1	14411			2983			15181		
3	4901	2.94	98.01	1255	2.38	79.24	5332	2.85	94.90
	(1396)			(1102)			(1454)		
6	2557	5.64	93.93	787	3.79	63.13	2763	5.49	91.58

[a] LCAP2 times given in parentheses

D. The two-electron integral derivative program

The HONDO program evaluates two-electron integral derivatives by two methods, the Rys quadrature method [23–27] and the method of Schlegel [28], the latter restricted to basis sets containing s and p functions and implemented in the GAUSSIAN 80 [29] and GAUSSIAN 82 [14] programs. Just like the integral programs, the integral derivative program, JKDER, loops over shell blocks, and the code may be easily parallelized by distributing this loop among the APs exactly as we have described for the integral programs.

Generally, the integral derivatives at a given geometry are needed only once. Accordingly, they need not be stored and may be summed directly into the gradient vector. Each AP evaluates the contribution of its own set of shell blocks to the skeleton gradient and the NAP contributions are added together and symmetrized [26, 27] on the host. It was not necessary to make any changes to the integral derivative code for execution on the APs.

4. Performance of the parallel program

To assess the efficiency of the parallel program we have carried out several calculations under benchmark conditions. The systems selected for study range from a small molecule with a near Hartree–Fock basis set to large molecules with

Table 2. Execution times, speedup factors, and efficiencies for the integral, SCF, and integral derivative programs in the calculations on VALINE (LCAP1, 3081 host; LCAP2, 3084 host)[a]

	Integral			SCF			Integral derivative		
NAP	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.
1	3813			1490			5468		
	(1106)			(1348)			(1509)		
3	1301	2.93	97.69	635	2.35	78.20	1845	2.96	98.78
	(387)	(2.86)	(95.26)	(621)	(2.17)	(72.36)	(532)	(2.84)	(94.55)
6	671	5.68	94.74	407	3.66	61.03	985	5.55	92.53

[a] LCAP2 times given in parentheses

Table 3. Execution times, speedup factors, and efficiencies for the integral, SCF, and integral derivative programs in the calculations on guanine (LCAP1, 3081 host)

	Integral			SCF			Integral derivative		
NAP	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.
1	1562			3694			1889		
2	792	1.97	98.59	2203	1.68	83.86	956	1.97	98.74
4	413	3.78	94.47	1458	2.53	63.36	491	3.85	96.14
6	308	5.07	84.46	1323	2.79	46.52	395	4.79	79.76
8	220	7.08	88.53	1112	3.32	41.54	261	7.25	90.59

Table 4. Execution times, speedup factors, and efficiencies for the integral, SCF, and integral derivative programs in the calculations on bicyclo(1,1,0)butane (LCAP1, 4381 host; LCAP2, 3084 host)[a,b]

	Integral[a]			SCF			Integral derivative		
NAP	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.
1	1014			808			3665		
	(263)			(580)			(1027)		
2	517	1.96	98.06	523	1.54	77.26	1843	1.99	99.42
	(138)			(344)			(521)		
4	268	3.79	94.65	374	2.16	54.03	933	3.93	98.23
6	201	5.05	84.23	331	2.44	40.75	684	5.35	89.23
8	147	6.88	88.97	313	2.58	32.23	478	7.66	95.71

[a] The Rys quadrature method was used throughout
[b] LCAP2 times given in parentheses

Table 5. Execution times, speedup factors, and efficiencies for the integral, SCF, and integral derivative programs in the calculations on water molecule (LCAP1, 3081 host; LCAP2, 3084 host)[a]

	Integral			SCF			Integral derivative		
NAP	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.
1	536			415			1371		
	(155)			(389)			(399)		
2	274	1.95	97.67	246	1.68	84.28	699	1.96	98.06
	(78)	(1.98)	(99.36)	(249)	(1.56)	(78.11)	(204)	(1.96)	(97.79)
3	196	2.72	90.98	198	2.10	69.94	487	2.82	93.91
	(67)	(2.31)	(77.11)	(210)	(1.85)	(61.74)	(152)	(2.65)	(87.50)
4	147	3.65	91.36	164	2.53	63.22	359	3.82	95.42
	(54)	(2.87)	(71.76)	(190)	(2.04)	(51.20)	(116)	(3.44)	(86.00)
5	123	4.35	87.12	157	2.65	53.05	305	4.50	90.04
	(50)	(3.10)	(62.00)	(177)	(2.19)	(44.00)	(103)	(3.87)	(77.48)
6	110	4.84	80.77	135	3.08	51.38	264	5.18	86.41

[a] LCAP2 times given in parentheses

Table 6. Execution times, speedup factors, and efficiencies for the integral, SCF, and integral derivative programs in the calculations on water dimer (LCAP1, 3081 host; LCAP2, 3084 host)[a]

	Integral			SCF			Integral derivative		
NAP	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.	T (sec)	Spdup.	Eff.
1	6659			5533			29189		
	(1900)			(4658)			(5769)		
2	3498	1.90	95.20	2943	1.88	93.99	14922	1.96	97.80
	(990)	(1.92)	(95.60)	(2469)	(1.87)	(94.32)	(3043)	(1.90)	(94.79)
3	2403	2.77	92.36	2084	2.65	88.49	10056	2.90	96.75
4	1807	3.68	92.10	1612	3.43	85.77	7617	3.83	95.79
	(523)	(3.63)	(90.80)	(1389)	(3.43)	(83.84)	(3043)	(3.71)	(92.69)
6	1270	5.24	87.38	1199	4.61	76.92	5171	5.64	94.07

[a] LCAP2 times given in parentheses

minimal basis sets. Our choice of systems should be sufficiently diverse to permit a reasonable assessment of the parallel program. The following six systems were used in the benchmarks:

1. Valine with a minimal basis set,
2. a crown ether with a minimal basis set,
3. guanine with the double-zeta valence 3-21G basis set,
4. bicyclo(1.1.0)butane with a double-zeta valence plus polarization basis set (C_{2v} symmetry),
5. the water molecule with a near Hartree-Fock limit basis set (C_{2v} symmetry),
6. the water dimer with a near Hartree-Fock limit basis set (C_s symmetry).

Full details of the basis sets, geometries, integral cutoffs, and convergence thresholds are available in a technical report [30]. Point group symmetry was exploited in the calculations on systems 4–6. Systems 1 and 2 are the 27 and 42 atom test cases used to test the efficiency of the parallel version of the IBMOL program [5]. In Tables 1–6 we present data obtained in the benchmarks. The efficiencies and speedups were calculated from the following formulae [4]:

$$\text{efficiency} = \frac{100\ (\text{host elapsed time for 1 AP})}{NAP\ (\text{host elapsed time for NAP APs})}$$

$$\text{speedup} = \frac{\text{host elapsed time for 1 AP}}{\text{host elapsed time for NAP APs}}$$

The general conclusions to be drawn from these results are that the integral and integral derivative programs execute in parallel with high efficiency but, largely because of an appreciable sequential component, the SCF program is less efficient. Comparing the timings for the integral and integral derivative programs on LCAP1 and LCAP2, we see that the execution times on LCAP2 are 3–4 times smaller than those on LCAP1, as anticipated [4]. For the SCF program, however, this is not the case. Although the FPS-264s execute floating point operations 3–4 times faster than the FPS-164s, the SCF program requires about the same time on

LCAP1 and LCAP2. This is because the SCF program is I/O bound [31], and the I/O rate of the FPS-264 machine is only about 20% better than for the FPS-164 for the FORTRAN I/O buffer size used in these runs.

A matter of great importance in parallel processing is the load balancing. So far we have always obtained reasonably good load balancing. We have implemented a scheme by which it may be possible to improve load balancing still further. This is to reorder the shells so that in the loop over shell blocks one evaluates first of all the (ss/ss) integrals, then the (ss/sp) integrals, and so on. The same scheme can be used in the integral derivative program. The fact that the 6 AP calculations on guanine and bicyclo(1.1.0)butane are less efficient than the 8 AP calculations can be attributed to imperfect load balancing in the 6 AP calculations. These 6 AP calculations would probably benefit from the reordering scheme.

6. Conclusion

Through the present work we have successfully parallelized the integral, Fock matrix formation, and integral derivative modules of the HONDO program. Our general method of parallelization should be of use in parallelizing other large FORTRAN codes. The integral and integral derivative programs were parallelized by the same technique, namely the distribution of the loop over shell blocks among the APs. The resulting parallel codes have been shown to execute with high efficiency with both minimal and near Hartree-Fock basis sets on small and large molecules alike. It is a simple matter to parallelize the code for computation of the second derivatives of the integrals in the same way. For reasons we have discussed, the SCF program as a whole does not execute very efficiently in parallel.

Recently, we have been using the parallel program to study some long chain polyenes and radicals. This work has involved geometry optimization and computation of harmonic vibrational frequencies [32]. These calculations would have been nearly impossible with a serial code.

References

1. Several such programs are available from the Quantum Chemistry Program Exchange (QCPE) at the University of Indiana, Bloomington, Indiana, USA, for a fee
2. Corongiu G, Detrich JH (1984) IBM Technical Report, KGN-1, March 31
3. Corongiu G, Detrich JH (1985) IBM J Res Dev 29:422
4. Clementi E, Chin S, Logan D (1986) "Chaire Francqui" Lecture Series: Part 9, IBM Kingston. See also Israeli Journal of Chemistry, in press
5. Clementi E, Corongiu G, Detrich JH, Khanmohammadbaigi H, Chin S, Domingo L, Laaksonen A, Nguyen HL (1984) IBM Technical Report, KGN-2, May 20
6. Clementi E, Corongiu G, Detrich JH, Khanmohammadbaigi H, Chin S, Domingo L, Laaksonen A, Nguyen HL (1984) In: Clementi E, Corongiu G, Sarma MH, Sarma RH (eds) Structure and motion: membranes, nucleic acids, and proteins. Adenine Press, New York
7. Clementi E, Corongiu G, Detrich JH, Khanmohammadbaigi H, Chin S, Domingo L, Laaksonen A, Nguyen HL (1985) Physica 131B:74
8. Clementi E (1985) J Phys Chem 89:4426 and references therein
9. Detrich JH, Corongiu G, Clementi E (1984) Int J Quantum Chem Symp 18:701

10. Detrich JH, Corongiu G, Clementi E (1984) Chem Phys Lett 112:426
11. Dupuis M, Spangler D, Wendoloski JJ (1980) NRCC Software Catalog., vol 1, Program No. QG01
12. Pulay P (1977) In: Schaefer HF III (ed) Applications of electronic structure theory. Plenum, New York
13. Schlegel HB, Binkley JS, Pople JA (1984) J Chem Phys 80:1976
14. Binkley JS, Frisch MJ, Raghavachari K, DeFrees DJ, Schlegel HB, Whiteside RA, Fluder G, Seeger R, Pople JA (1983) GAUSSIAN 82, Release A, Carnegie-Mellon University, Pittsburgh, USA
15. Roothaan CCJ (1951) Rev Mod Phys 23:69
16. McWeeny R, Sutcliffe BT (1969) Methods of molecular quantum mechanics. Academic Press, New York
17. Szabo A, Ostlund NS (1983) Modern quantum chemistry: an introduction to advanced electronic structure theory. Macmillan, London
18. Goddard JD, Handy NC, Schaefer III HF (1979) J Chem Phys 71:1525
19. Bobrowicz FW, Goddard III WA (1977) In: Schaefer III HF (ed) Methods of electronic structure theory. Plenum, New York
20. Chin S, Domingo L, Carnevali A, Caltabiano R, Detrich JH (1985) IBM Technical Report, KGN-42, November 25
21. Pople JA, Hehre WJ (1978) J Comput Phys 27:161
22. Hehre WJ, Lathan WA, Ditchfield R, Newton MD, Pople JA (1970) GAUSSIAN 70, Program 236, QCPE, University of Indiana, Bloomington, Indiana
23. Dupuis M, Rys J, King HF (1976) J Chem Phys 65:111
24. King HF, Dupuis M (1976) J Comput Phys 21:144
25. Rys J, Dupuis M, King HF (1983) J Comput Chem 4:154
26. Dupuis M, King HF (1985) IBM Technical Report, KGN-26, August 26
27. Dupuis M, King HF (1986) In: Jorgensen P, Simons J, Reidel D (eds) Geometrical derivatives of energy surfaces and molecular properties
28. Schlegel HB (1982) J Chem Phys 77:3676
29. Binkley JS, Whiteside RA, Krishnan R, Seeger R, DeFrees DJ, Schlegel HB, Topiol S, Kahn LR, Pople JA (1981) GAUSSIAN 80, QCPE 13:406
30. Watts JD, Dupuis M, Villar HO (1986) IBM Technical Report, KGN-78, August 29
31. Dupuis M, Watts JD (1986) IBM Technical Report, KGN-69, July 28
32. Villar HO, Dupuis M, Watts JD, Hurst GJB, Clementi E: to be published