An improved criterion for evaluating the efficiency of two-electron integral algorithms

Michael J. Frisch
Lorentzian, Inc., 127 Washington Avenue, North Haven, CT 06473, USA

Benny G. Johnson, Peter M.W. Gill
Department of Chemistry, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Douglas J. Fox
Gaussian, Inc., 4415 Fifth Avenue, Pittsburgh, PA 15213, USA

and

Ross H. Nobes
Australian National University Supercomputer Facility, GPO Box 4, Canberra ACT 2601, Australia

Received 31 December 1992; in final form 15 February 1993

We present a general criterion for theoretical performance assessment of algorithms for two-electron integral computation which is appropriate for most modern computers. The new prescription is to minimize the total number of memory references in the algorithm, as opposed to the traditional approach of minimizing the total number of floating-point operations. CPU timings on a range of machines demonstrate that memory operations are better correlated to machine cycles than are floating-point operations.

1. Introduction

Efficient computation of the ubiquitous two-electron repulsion integral (ERI) in ab initio quantum chemical calculations has long been a topic which has received much attention in the research literature [1–14]. In Hartree–Fock calculations the number of ERIs which must be computed is formally $\mathcal{O}(N^4)$, where N is the number of basis functions. For large calculations where the disk space required to store the ERIs is prohibitive, a "direct" method, in which the ERIs are recomputed each time they are needed, must be used. As ERI evaluation is the dominant step in such a procedure, in order for direct methods to be feasible, highly optimized algorithms for ERI generation are required.

In addressing the issue of designing an ERI algorithm for which the computer implementation will be optimal, it is obvious that the ultimate criterion is that the total number of machine cycles required must be minimized. However, optimizing machine cycles is undesirable because the solution is highly machine dependent, which is contrary to the preferable notion of designing a single algorithm which performs efficiently on all machines of interest. Accordingly, we seek a general cost parameter which gives a good approximation to machine cycles on most machines.

In order to take maximum advantage of vector architectures, it is desirable that as much as possible of the computationally intensive work in the algorithm be performed inside iterative loops which simply add and multiply elements of long arrays (vectors), and we shall focus our attention on these sorts of constructs. Traditionally, the theoretical measure of the cost of an algorithm has been the total number of

floating-point operations, or flops (additions, subtractions, multiplications, and divisions), and algorithms requiring the fewest flops were considered optimal [15]. However, this does not give the best indication of machine cycles on many modern computers which are capable of loading or storing a floating-point number from or to memory, completing a multiplication, and completing an addition in one machine cycle. In this case, the cost C in machine cycles of executing a program statement performing only these types of operations is given by

$$C=\mathrm{MAX}(L+S, M, A)\,, \qquad (1)$$

where $L+S$ is the total number of loads and stores, M is the number of multiplications, and A is the number of additions in the statement.

We therefore propose that the appropriate cost parameter for theoretical performance assessment of ERI algorithms is C. This cost parameter is exact for machines of the type described above (neglecting the comparatively few cycles required for tasks such as maintaining loop counters), whereas the number of flops ($=M+A$) is a poorer measure of CPU time since memory references are ignored. Indeed, since multiplication and addition are binary operations, for practical purposes it will almost always be the case that $L+S$ is greater than M or A, and memory operations, not flops, will be the determining factor for CPU time consumed on such machines. Henceforth, we shall take C to be equal to $L+S$, the total number of memory operations, or mops. In the remainder of this paper, we establish the utility of mops as a cost parameter, by demonstrating that machine cycles are better correlated to mops than to flops over a wide variety of machines. In subsequent Letters [16,17], it will be illustrated how the minimum-mops optimality criterion is applied to representative intermediate steps in an ERI algorithm, with the results compared with currently used methods.

Before proceeding, we note some other relevant work in this area. The importance of mops has been recognized in the computer literature by, for example, Dongarra [18], who writes: "Algorithm performance can be dominated by the amount of memory traffic rather than by the number of floating-point operations involved". Other consideration of theoretical performance metrics involves two-parameter representations [19,20]; however, here we use only the single parameter mops, as this avoids the additional complexity in algorithm optimization required by consideration of multiple parameters, and also because considerable improvement over flops can be obtained simply by using an improved single-parameter metric. It should be noted, however, that while the rationale behind advocating mops is appropriate for a single-processor vector architecture model, when designing algorithms for parallel architectures more sophisticated considerations which are highly machine-dependent are necessary, as has been pointed out, for example, by Colvin, Whiteside and Schaefer [21].

2. Correlation of machine cycles and mops

In table 1 are listed several abstract formulae which represent recurrence relation transformations commonly appearing in ERI algorithms [11,14]. For each, the cost in flops and mops is given in the first column. The determination of the flop costs is straightforward, as it is simply the total number of additions and multiplications in the formula. In determining the mop costs, real constants, denoted by lower-case letters, are loop invariants and should not be counted, and it is assumed that any variable appearing more than once on the right-hand side is loaded into a storage register where it may subsequently be rapidly accessed as needed, and thus each distinct variable is counted only once. The remaining columns give the average number of machine cycles elapsed per operation performed for a variety of machines.

We find that the average number of machine cycles per mop is more constant over the given set of formulae than the number of cycles per flop for most of the machines in table 1, excepting the VAX station 3100, for which neither flops nor mops give a good correlation with machine cycles. This is expected since the VAX station 3100 is a scalar sequential machine, i.e. its architecture is not at all similar to that implying eq. (1). A more appropriate definition of C for this machine would be $L+M+A$.

The cycles per mop values on the Cray Y-MP/832 and Fujitsu VP-100 for the next-to-last formula are relatively high, and merit elaboration. These machines are capable of completing up to two mops, an

Table 1
Theoretical costs and machine cycles per operation for various formulae

Formula	Cost	Machine cycles/operation [a]				
		VAX station 3100	Alliant FX/8	Multiflow Trace 14/300	Cray Y-MP/832	Fujitsu VP-100
$W=W+A$						
flops	1	41	7.0	1.2	1.6	1.9
mops	3	14	2.3	0.39	0.54	0.63
$W=W+A+B$						
flops	2	31	4.9	0.78	1.1	1.2
mops	4	15	2.5	0.39	0.57	0.60
$W=W+A+B+C$						
flops	3	29	4.1	0.66	1.1	1.1
mops	5	17	2.5	0.40	0.66	0.64
$W=A+XB$						
flops	2	34	4.7	0.78	1.2	1.2
mops	4	17	2.3	0.39	0.59	0.58
$W=A+XB+YC$						
flops	4	35	4.2	0.65	1.0	0.86
mops	6	23	2.8	0.43	0.67	0.57
$W=A+X(B+YC)+YD$						
flops	6	27	3.2	0.49	0.75	0.69
mops	7	23	2.8	0.42	0.64	0.59
$W=XA+YB+mC+nD$						
flops	7	24	2.9	0.52	0.71	0.60
mops	7	24	2.9	0.52	0.71	0.60
$W=A+X[mB+X(nC+XD)]$						
flops	8	22	2.0	0.44	0.67	0.66
mops	6	29	2.7	0.58	0.89	0.88
$W=A+X[B+Y(C+ZD)+ZE]+Y(F+ZG)+ZH$						
flops	14	25	2.9	0.46	0.65	0.62
mops	12	29	3.3	0.54	0.75	0.72

[a] Each formula was coded inside a FORTRAN DO loop of length 1000, which was compiled at high optimization and executed sufficiently many times to accumulate one CPU second on each machine (one processor, vector mode where applicable). The ratios were calculated from these timings.

add and a multiply in one cycle, so eq. (1) is not strictly valid for them. For the formula in question, these machines will actually be rate-limited by multiplications, not mops. This shows that the minimum-mops criterion should not be applied indiscriminately, but the fact that the ratio is nearly constant for the rest of the set of formulae illustrates the usefulness of the simple cost parameter mops: it is successful in approximating machine cycles even for some architectures where it is not the most appropriate cost parameter possible.

Therefore, it is established that optimizing an algorithm with respect to mops is an excellent approximation to optimizing machine cycles, and that this is superior to optimizing the total number of flops. Furthermore, the observed constant ratios assert the validity of the assumption implicit in the counting scheme used, namely that an adequate number of registers are available so that a sufficiently resourceful compiler need load each distinct variable only once. Since the number of registers as well as compiler skill is finite, this assumption cannot continue to hold as formulae become more and more complicated, and we notice a slight increase in cycles/mop on most machines for the more complicated formulae at the bottom of table 1, but it is important to note that the assumption does generally hold for the formulae studied here, most of which are actually implemented in the PRISM ERI algorithm [11,14]. Certain of these formulae are dis-

cussed in detail in a following Letter [16].

3. Concluding remarks

We have demonstrated that total mops is a better cost parameter for theoretical performance assessment of ERI algorithms than total flops due to the better correlation of machine cycles to mops on most modern computers. Therefore, when designing ERI algorithms, the total number of mops should be optimized. The minimum-mops optimality criterion can be applied to both the basic types of individual steps comprising most ERI algorithms, contraction and transformation, to yield improvements in CPU time required [16,17]. It should be stressed that such improvements are obtained by methods of reducing total mops which do not necessarily correspond to minimizing or even reducing the total number of flops; indeed, it is sometimes the case that the minimum-mops implementation of a particular task is much more expensive in terms of flops than an implementation which is inferior in performance [16].

Finally, we point out that although the use of mops as a theoretical cost measure is advocated here in the context of optimizing ERI algorithms, the utility of such optimization procedures is not limited to this area; it is expected that the efficiency of algorithms in other areas of computational science which are currently optimized with respect to flops can be improved by applying the minimum-mops criterion.

Acknowledgement

This work was supported by the National Science Foundation under Grant CHEM-8918623. Professor John Pople is kindly acknowledged for his comments on this manuscript. We are indebted to the referee for bringing refs. [18–21] to our attention. BGJ thanks the Mellon College of Science for a Graduate Fellowship.

References

[1] H.F. King and M. Dupuis, J. Comput. Phys. 21 (1976) 144.
[2] L.E. McMurchie and E.R. Davidson, J. Comput. Phys. 26 (1978) 218.
[3] J. Rys, M. Dupuis and H.F. King, J. Comput. Chem. 4 (1983) 154.
[4] S. Obara and A. Saika, J. Chem. Phys. 84 (1986) 3963.
[5] M. Head-Gordon and J.A. Pople, J. Chem. Phys. 89 (1988) 5777.
[6] P.M.W. Gill, M. Head-Gordon and J.A. Pople, Intern. J. Quantum Chem. Symp. 23 (1989) 269.
[7] P.M.W. Gill, M. Head-Gordon and J.A. Pople, J. Phys. Chem. 94 (1990) 5564.
[8] T.P. Hamilton and H.F. Schaefer III, Chem. Phys. 150 (1991) 163.
[9] R. Lindh, U. Ryu and B. Liu, J. Chem. Phys. 95 (1991) 5889.
[10] I. Panas, Chem. Phys. Letters 184 (1991) 86.
[11] P.M.W. Gill and J.A. Pople, Intern. J. Quantum Chem. 40 (1991) 753.
[12] P.M.W. Gill, B.G. Johnson and J.A. Pople, Intern. J. Quantum Chem. 40 (1991) 745.
[13] B.G. Johnson, P.M.W. Gill and J.A. Pople, Intern. J. Quantum Chem. 40 (1991) 809.
[14] P.M.W. Gill, Advan. Quantum Chem., in press.
[15] D. Hegarty and G. Van der Velde, Intern. J. Quantum Chem. 23 (1983) 1135.
[16] B.G. Johnson, P.M.W. Gill and J.A. Pople, Chem. Phys. Letters 206 (1993) 229.
[17] B.G. Johnson, P.M.W. Gill, J.A. Pople and D.J. Fox, Chem. Phys. Letters 206 (1993) 239.
[18] J.J. Dongarra, Technical Memorandum No. 23, Mathematics and Computer Science Division, Argonne National Laboratory (1988).
[19] R.W. Hockney and C.R. Jesshope, Parallel Computers (Adam Hilger, Bristol, 1981).
[20] R.W. Hockney, Supercomputer 48 (1992) 9.
[21] M.E. Colvin, R.A. Whiteside and H.F. Schaefer III, Methods Comput. Chem. 3 (1989) 167.