

Massively Parallel Full Configuration Interaction. Benchmark Electronic Structure Calculations on the Intel Touchstone Delta.

Robert J. Harrison

Theoretical Chemistry Group, Chemistry Division,
Argonne National Laboratory, Argonne IL 60439, U.S.A.

Eric A. Stahlberg¹

Molecular Science Research Center,
Pacific Northwest Laboratories,
Richland, WA 99352, U.S.A.

We describe an implementation of the benchmark *ab initio* electronic structure full configuration interaction model on the Intel Touchstone Delta. Its performance is demonstrated with several calculations, the largest of which (95 million configurations, 418 million determinants) is the largest full-CI calculation yet completed. The feasibility of calculations with over one billion configurations is discussed.

A sustained computation rate in excess of 4 GFLOP/s on 512 processors is achieved, with an average aggregate communication rate of 155 Mbytes/s. Data-compression techniques and a modified diagonalization method were required to minimize I/O. The object-oriented design has increased portability and provides the distinction between local and non-local data essential for use of a distributed-data model.

The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. W-31-109-ENG-38. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

¹Current address, Theoretical Chemistry Group, Chemistry Division, Argonne National Laboratory, Argonne IL 60439, U.S.A.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

1 Introduction

Full-configuration interaction (full-CI) provides the *exact* solution of the electronic Schrödinger equation within the initial algebraic approximation adopted by all mainstream *ab initio* electronic structure methods and is the result that all other models strive to approximate. The only errors present in a full-CI result derive from either the underlying finite one-particle basis set or approximations in the non-relativistic, Born-Oppenheimer Hamiltonian. The ability to compute full-CI wavefunctions thus confers the ability to adjudicate between all approximate methods and, by comparison with experiment, permits assessment of deficiencies in the one-particle basis set and the Hamiltonian approximations.

The full-CI problem scales approximately exponentially with the number of electrons and is algorithmically expressed as the extraction of the lowest eigenvalue and corresponding vector from a large sparse matrix. It is only in recent years that new algorithms [1, 2, 3, 4, 5, 6] and the very largest computers have enabled full-CI calculations to be performed on systems sufficiently large to provide meaningful information. Full-CI results obtained since the advent of large-memory vector supercomputers [7, 8, 9, 10, 11, 12] have provided insights into the applicability and convergence of many approximate methods and have led to a complete redesign of the one-particle basis sets for high-accuracy computation [13, 14]. As a result of these efforts, present-day high-accuracy calculations must always be documented by evidence demonstrating convergence to the result obtained by full-CI.

RECEIVED
OCT 14 1994
OSTI

The need exists for further exact benchmarks on both energetics and properties, particularly on systems such as small metallic clusters which are problematic to study using common approximate methods. There have been abortive attempts at performing much larger computations, but the computational resources available (cpu, memory, disk-space and I/O bandwidth) have proven inadequate [15]. The Touchstone Delta provides capabilities well surpassing at least some of these resource limitations, and with aggressive algorithmic development can open up a new set of systems for scrutiny at the full-CI level of theory.

We describe below an implementation on the Touchstone Delta of the most efficient full-CI algorithm known. This fully spin-adapted code is only just commencing production, yet has already permitted interactive execution of calculations in only minutes that would previously have taken days. A design emphasis on encapsulation and data hiding have given rise to an object-oriented structure. This has led to valuable code reuse and, most significantly, a high level of portability. For instance, an object implemented on the Delta as an asynchronously-accessed distributed data structure has been implemented on our Alliant FX2800 with shared memory and locks. This implementation is hidden from the main body of the application, which remains the same in both environments.

1.1 The Intel Touchstone Delta Field Prototype

The Touchstone Delta [16] is a 16x32 2-D mesh of Intel i860 [17] processors (40 Mhz). On the edges of the mesh are 32 nodes (Intel 386 processors [17]) devoted to I/O

and several service nodes responsible for networking and interactive access. Each i860 processor has 16 Mbyte of local memory, of which about 13 Mbyte are available to applications. Processors communicate via explicit message passing with the standard Intel library [18]. The peak communication speed is 25 Mbytes/s but work-arounds to routing bugs and flow control reduce this to approximately 7 Mbytes/s. Bypassing flow control increases the available bandwidth to approximately 14 Mbytes/s [19]. Hardware fixes, planned for May 1992, will eliminate the routing-bugs and restore full hardware bandwidth. Communication latency is highly sensitive to context and system load, with representative figures being 50-380 μ s [19].

2 Full-CI theory and algorithm in brief

Previous large full-CI calculations have used a cpu and memory intensive determinantal basis due to its computational simplicity [1, 3, 6, 4, 5]. In distinction, the algorithm employed here (due to Duch [20]) utilizes a fully spin-adapted² basis and achieves a reduction in both memory and cpu requirements.

The iterative diagonalization [21] proceeds via multiplication of a trial vector (c) by the Hamiltonian matrix (H). In the following we consider in detail just the expensive two-electron contribution to the matrix-vector product [20]. Factorization of the Hamiltonian operator [2] in second-quantized form and projection onto the $N-2$ electron subspace [4, 5] (labeled K below) permits the majority of floating-point

²Spin-adapted implying that full use is made of symmetries arising from the coupling of electronic spins.

work for the matrix-vector product to be recast as a large matrix-matrix product. Specifically, the two-electron contribution to the matrix-vector product (σ) is written [20] as

$$\begin{aligned}
 \sigma_I &= \sum_J H_{IJ}^{(2)} c_J \\
 &= \sum_{Kik} \langle I | E_{ia} E_{kb} | K \rangle \sum_{jl} \langle ik | jl \rangle \sum_J \langle K | E_{aj} E_{bl} | J \rangle c_J \\
 &= \sum_{Kik} \langle I | E_{ia} E_{kb} | K \rangle \sum_{jl} \langle ik | jl \rangle D_{jl,K} \\
 &= \sum_{Kik} \langle I | E_{ia} E_{kb} | K \rangle E_{ik,K}
 \end{aligned} \tag{1}$$

The dense matrix D is constructed through contraction of the CI vector with the very sparse matrix elements of the unitary group operators E_{bl} [22]. The sums over the orbital index pairs (ik and jl) are constrained to the upper triangle by exploiting a special spin-coupling scheme [20]. The D matrix is then multiplied by the two-electron integral matrix to form the matrix E . The scattering of E into the product vector (σ) proceeds as the reverse of the process by which D was constructed.

The N and $N-2$ electron functions are lexically addressed from a common graphical representation and this gives rise to several valuable properties. Notably, interactions of neighboring intermediate states are likely to require the same CI coefficients, or at least CI coefficients that are “close-by”. Only when the graphical representation of an intermediate state changes significantly will completely different segments of the CI vector be addressed. These properties permit the CI vector to be implemented efficiently as a distributed data structure, extended to paging from disk (with caching) when required. By gathering only the unique CI coefficients that are

required for a set of neighboring intermediate states, the number of references to the CI vector is reduced, thereby increasing efficiency.

Let N_{CI} be the length of the full-CI expansion (typically $10^7 - 10^9$), n the number of electrons (where 16 is a practical upper limit), m the number of orbitals (10-100) and, as is the case for most applications, assume that $m \gg n$. Then, ignoring use of spatial symmetry, the following statements may be made about the work required at each stage.

- Each CI coefficient contributes to approximately $n^2/2$ elements of the intermediate matrix D , which is of dimension $N_{CI}n^2/2$. The number of operations to form D is proportional to its dimension, with a prefactor that is larger in a spin-adapted basis than with determinants. The same amount of work is required to scatter E back into the result (σ).
- The number of floating-point operations in the matrix multiplication forming E is approximately $N_{CI}n^2m^2/2$ (for singlet states, also taking into account the extra symmetry from the spin-coupling scheme mentioned above).

In practice, profiling a modest example on a single i860 (Alliant FX2800), the matrix multiplication (using DGEMM, achieving 35 Mflop) takes about the same amount of time as all the rest of the computation, which is a mixture of integer and floating-point work. This is not expected to change significantly for larger calculations.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

3 Full-CI in parallel

Our objectives for this program were to:

- enable much larger full-CI benchmarks of chemically significant systems;
- perform efficiently on the Delta and be scalable to larger massively-parallel machines;
- investigate the applicability of an object-oriented programming style to electronic structure codes, especially for the case of parallel machines;
- maximize portability of the program between serial or parallel shared-memory and distributed-memory architectures; and
- incorporate our experiences in the design of the next generation of *ab initio* correlated programs.

The next sections describe how we achieved some of these goals — the rest is still in progress.

3.1 Work decomposition

The two-electron interaction described above dominates the computation and we shall focus on this aspect. The one-electron interactions proceed in an analogous fashion, but are much less significant.

The intermediate matrices D and E easily become too large to hold in available memory and therefore must be constructed and processed in sections. This partitioning serves as the foundation for the parallel algorithm. A parallel task is specified by a range of intermediate (K) states (and associated information) which specify a unique block of D . This decomposition is precomputed (in parallel), typically resulting in 2000-60000 blocks or tasks. The selection of the block size controlling the partitioning of D forces a compromise between several important considerations.

- Task granularity and load balancing.
- Memory available per node (a typical block size is 200,000 doubles).
- Reduction in communication — the larger the block size, the greater the reuse of data. The reuse is in the range 2–5 with typical block sizes.

3.2 Data decomposition

The largest data structures that must be retained are the CI trial (c) and result (σ) vectors (including associated addressing information). These vectors are of sufficient size that the sum of available memory for all processors must be utilized in order to retain the vectors in memory. This necessitates the creation of a blocked structure spanning the available processors. When even the aggregate memory is of insufficient size, the access pattern described in Section 2 is expected to allow these blocks to be paged efficiently from disk (this has yet to be tested). Both vectors are similarly distributed between all nodes. The selection of the CI vector block

size requires a compromise between the average message size, even distribution of the communication load across all nodes, memory usage and the amount of caching required for successful paging from disk.

The object-oriented design and the desire for portability imply that the implementation details of the vectors be hidden, with the application only accessing the “object” by its associated methods. The blocking of the CI vector is again performed in parallel, reusing the same code which blocked the $N - 2$ (and $N - 1$) intermediate states.

3.3 Implementation

Our goals here are to express the object-oriented design and use this to maintain portability while mapping the objects efficiently onto the Delta. To maintain scalability, and to enable very large computations to be performed, *all* significant data structures and operations must be distributed. The program is implemented in C with calls to the BLAS (levels 1 and 3) in numerically intensive sections. C was chosen rather than C++ because the latter is not yet as widely available or standardized, and to avoid the large learning curve associated with C++. C permits encapsulation and data hiding but does not support this with the static-type checking, polymorphism, operator overloading and class inheritance available in C++.

We focus now on the key object, the CI vector. This object must support requests to gather/scatter sets of coefficients with appropriate mutual exclusion and deadlock avoidance. A node cannot anticipate when another node will need

access to the data it is holding. To handle such requests, asynchronous handlers (using the N/X routine `hrecv()`) are established. In the case of the scatter operation, mutual exclusion is enforced (using `masktrap()`) to render this operation effectively atomic. Since it is also possible for a cycle of processors to request simultaneously for a service to be performed (potentially resulting in deadlock), local handlers are masked if the target remote node is of greater index than the local node [23].

In order to efficiently utilize all available processors dynamic load-balancing is required for the following reasons.

- The time taken to perform tasks varies by at least a factor of two.
- There may exist temporary communication “hot-spots”. Randomization of the tasks would effectively minimize this but would preclude an effective caching algorithm for paging from disk.
- In a multiuser shared-memory environment, processes may be swapped out or scheduled unfavorably. Dynamic load-balancing minimizes the impact of the run-time environment.

Presently, dynamic load-balancing is supported by a shared counter (again implemented with an `hrecv()` handler) that may be incremented by arbitrary amounts to control granularity.

Communication time is minimized by fully exploiting the reuse of data within a task and by use of N/X “force-types” which effectively double the available bandwidth, bypassing normal flow-control [19, 24].

3.4 Performance estimation

Assuming a uniform distribution of the CI vector across the P processors, the amount of data that each node will both send and receive in using the distributed c and σ vectors is approximately (in double precision words) $N_{CI}n^2/\alpha P$, where α reflects reuse of data.

The effective bandwidth per node for P processors communicating randomly on a square mesh is S/\sqrt{P} , where S is the link speed [24]. A model program performing random communications via `hrecv()` and no computation demonstrated such scaling with an aggregate bandwidth of 308 Mbytes/s on 512 nodes. This corresponds to an effective link speed of $S=13.6$ Mbytes/s (using force-types and running the slow “router-bug-fix” kernel).

The total communication time in seconds is then estimated as

$$t_{comm}(P) = \frac{8N_{CI}n^2}{\alpha S\sqrt{P}} \quad (2)$$

This assumes that latency is unimportant (i.e. large messages) and a relatively even distribution of communications across the machine. The latter is established by wrapping the CI vector several times around the machine.

Assuming an even work distribution, the time taken by useful computation per node is approximately

$$t_{cpu}(P) = \frac{N_{CI}n^2m^2}{PM} \quad (3)$$

where M is the effective computation rate (estimated as 20 Mflop/s). Thus, a crude

estimate of the expected efficiency is

$$\begin{aligned}\epsilon &= \frac{t_{comm}(1) + t_{cpu}(1)}{P(t_{comm}(P) + t_{cpu}(P))} \\ &= \frac{1 + m^2\alpha S/(8M)}{\sqrt{P} + m^2\alpha S/(8M)}\end{aligned}\quad (4)$$

Thus, if $m^2\alpha S/(8M) \gg \sqrt{P}$ good efficiency is expected. Substituting the representative values $m = 20$, $\alpha = 3$, $S = 13$ Mbytes/s, $M = 20$ MFLOP/s, we arrive at

$$\epsilon = \frac{98.5}{\sqrt{P} + 97.5}\quad (5)$$

This crude estimate gives us some confidence that the program will perform well on the available number of processors. A much more detailed performance model is under development which will take into account exact operation counts, spatial symmetry, latency, and empirical evidence of sparsity, communication patterns and load balancing.

3.5 Delta specific modifications

Several hardware and software aspects make the Delta more challenging than other environments and must be confronted for successful program development and efficient execution.

The very limited bandwidth to disk is the single largest problem. The Davidson diagonalization algorithm [21] expands the solution in the iterative subspace, and requires storage and reprocessing of the trial vectors and corresponding matrix-vector

products. This is a complete disaster on the Delta. Several major modifications were made to significantly reduce, and for small cases completely eliminate, I/O.

- Davidson's algorithm [21] was modified to use an expansion space of just three vectors and constant denominators formed from orbital energies. The matrix-vector products no longer need to be stored on disk and the amount of I/O is reduced by a factor of approximately three. Eight reads and three writes of full vectors are now required for two Davidson updates. The price for this I/O reduction is doing three matrix-vector products for every two Davidson updates. On 512 nodes it seems that a matrix-vector product is about as fast as reading a vector from disk, so an advantage is realized.
- The current solution vector must be kept to full-precision, while the two update vectors need only be retained to sufficient precision to avoid degrading convergence [25]. In practice, the two update vectors are compressed from arrays of doubles to arrays of bytes. The compression algorithm preserves the symmetry between $\pm x$ and thus maps small numbers to zero [25]. The I/O is now reduced to 3.625 reads and 1.25 writes of a full vector every two updates.
- Small calculations (less than 50 million configurations) buffer all I/O, the only I/O operation being to write out the final solution.
- To address files greater than $2^{31} - 1$ bytes, an extended addressing mechanism must be used. Preallocation of files also greatly speeds subsequent write operations.

The last three improvements required modification of only the implementation of the CI vector — the application was unchanged.

The system-provided `malloc()/free()` dynamic memory-allocation routines have great overhead associated with them, no error checking and no diagnostic or performance tools. Much faster, more robust and functional equivalents were implemented and were found essential in debugging and tuning this program in which efficient memory usage is critical.

The compilers, while robust, still lack performance, especially when much logic or indirection is used. The sparse matrix-vector product routines used to multiply the CI vector by the unitary-group coupling coefficients were coded in i860 assembler. These now achieve 8 MFLOP/s on relatively small matrices and make full use of the sparsity in both the matrices and the vector.

3.6 Performance statistics

We analyze the performance of three small examples to demonstrate the scalability and general performance of the program, and then examine a larger problem that is more representative of our target applications. The relevant dimensions are given in Table 1, and the corresponding energies in Table 2. Figures 1 and 2 demonstrate scaling for the two smaller examples which could be run on smaller submeshes. Table 3 contains detailed timing statistics, including sustained computation rates. The computation rates are evaluated as the number of operations spent in just 2-electron matrix-multiply operation divided by the total global time taken to perform

a matrix-vector product. The reported computation rates are thus rigorous *lower bounds* to the true rates, which are estimated to be up to 50% greater. A more detailed performance model and additional run-time statistics are required for more accurate information. Table four summarizes the communication performed by each calculation, including the total data transferred, the sustained communication rate and the total number of message sent. Figures 3-6 provide a detailed analysis of the communications performed in the smaller neon calculation on 144 nodes and are discussed in that section.

3.6.1 The methyl radical

A full-CI calculation was performed on CH_3 with the 1s core frozen as a canonical MCSCF orbital, in a cc-pVDZ basis set [14] at a C_s geometry representing a slight displacement from equilibrium ($r_{CH} = 1.07900\text{\AA}$, in-plane $\angle \text{HCH} = 138.513^\circ$, out-of-plane $\angle \text{H-CH}_2 = 19.436^\circ$). This is part of a sequence of calculations performed to determine the full-CI harmonic vibrational frequencies to benchmark multireference single and double excitation CI results [26]. This small problem demonstrated reasonable scaling (Figure 1), and sustained in excess of 4.1 GFLOP/s on 512 processors. The energy converged to six decimal places in 10 iterations of the modified Davidson procedure.

3.6.2 The neon atom in two basis sets

For comparison with previous results, to demonstrate program correctness and performance, we reproduced the neon atom results of Olsen *et al* [15] in their 5s3p2d1f and 6s4p3d basis sets. The previous energies were recovered to all reported figures, converging to six decimal places in eight iterations of the modified Davidson procedure. We are presently only able to use D_{2h} symmetry, rather than the $D_{\infty h}$ symmetry used previously [6]. This implies that we are doing twice as much work. Olsen *et al* [15] report a time of 6390s per matrix-vector product for the 6s4p3d basis, on an IBM 3090 with a single vector unit. Since their algorithm [6] differs from ours it is not meaningful to compare timings directly.

For the 6s4p3d problem our code sustained in excess of 2 GFLOP/s. This is less than the CH_3 example, as the use of additional spatial symmetry reduces the number of orbitals in a symmetry block, which according to the simple performance model should degrade performance. The size of integral blocks is also reduced (which degrades the performance of the matrix operation) and decreases the number of floating-point operations in the matrix multiplications.

The timings for the 5s3p2d1f basis on 144 processes exhibit an anomaly. The second iteration took 46s longer than the average of all the other iterations, and is responsible for the large variance reported in Table 3 and shown in Figure 2. This is the only such occurrence out of many hundreds of iterations we have now run and we have not yet had opportunity to repeat the computation to see if this effect is reproducible. We have just instrumented the program with the Argonne

performance analysis tools [27] with one objective being to provide a visual display of events in this iteration.

A detailed analysis of interprocess communication in the Ne 5s3p2d1f example running on 144 nodes is given in Figures 3-6 for a single matrix-vector product. Figure 3 displays the number of interrupts generated on each node in servicing the distributed CI vector, while Figure 4 displays the actual amount of data sent. In this example, the 509 blocks generated with a 50,000 word blocking of the CI vector wrap around the machine approximately 3.5 times. This incomplete wrapping of the CI vector is seen in Figures 3 and 4 with the shift downward of handler interrupts apparent at node 76. The wide range of values in Figures 3 and 4 is accounted for by the unequal wrapping and the existence of several small CI vector blocks resulting from the parallel blocking scheme.

Figure 5 shows the number of requests each node made for remote CI vector coefficients, and Figure 6 the corresponding volume of data. The well balanced requests for data (compared with the handler interrupts and handler data traffic, Figures 3 and 4) results from the dynamic load balancing employed in the calculation. The slight shift upward of both the remote node accesses and the remote data requested also at node 76 is again explained with reference to the incomplete wrapping of the CI vector. Those nodes with fewer CI-vector blocks are able to process more of the parallel tasks (resulting in a slight increase in data requests) simply due to the fact that fewer handler interrupts must be serviced.

3.6.3 Methane

Our largest calculation to date, is also the largest full-CI calculation successfully completed. Methane ($r = 1.085600\text{\AA}$) in the same cc-pVDZ basis set [14] used for CH_3 . The calculation was run in a C_{2v} subgroup of T_d , again with a frozen canonical SCF core orbital. We are in the process of computing the full-CI equilibrium geometry, harmonic vibrational frequencies and polarizabilities for methane in this basis set. Combined with the CH_3 results and similar calculations on CH_2 , CH and the carbon atom, we shall be able to provide full-CI results for the main energetics and properties of this entire sequence of molecules, which vary greatly in their electronic structure. Six decimal places of the full-CI energy were recovered in eight iterations of the Davidson process. This computation sustained in excess of 4.0 GFLOP/s if I/O is excluded from the timings. Including I/O time, the performance drops to just 1.7 GFLOP/s.

4 Conclusions

We have an implementation of the full-CI method that scales well on the Intel Touchstone Delta and have demonstrated its performance with several benchmark calculations. The largest of these calculations (95 million CSF, 418 million determinants) is the largest full-CI calculation completed to date. Our current capabilities are already valuable in a making a new set of full-CI benchmarks with up to 250 million configurations routine. Straightforward improvements, which are underway,

should enable benchmark calculations with up to 2,000 million configurations to be performed on the Delta, albeit with significant effort. Calculations of such size are needed to provide FCI benchmarks for highly-correlated systems in which size extensive effects are substantial, such as metallic clusters, and to benchmark high-order response properties in appropriately large basis sets.

The emphasis on encapsulation and data hiding has paid off greatly, with demonstrable improvements in portability, code reuse, productivity, and functionality. We believe that this has significant implications for the design of many new programs in computational chemistry, and for the languages and computer science that upcoming computational scientists should be exposed to. The data hiding also emphasized the fundamental distinction between local and nonlocal data. This has permitted the use of a distributed-data model, avoiding use of explicit message passing at the application level.

It must be recalled that the Touchstone Delta is a prototype. While it is potentially a supercomputer applicable to a wide range of applications, it is constrained to a much smaller class of applications by several major flaws. These include the appallingly low I/O rate to disk, poorly designed system software, high message-passing latency, low communication bandwidth, and low amount of memory per node. Some of these issues are driven by cost (e.g. memory size) or technology (e.g. memory, communication bandwidth). For the next generation of massively-parallel machines to prove widely applicable vendors must address these issues very aggressively.

5 Acknowledgments

Work by RJH was supported by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Chemical Sciences, under contract W-31-109-ENG-38. Work by EAS was performed under the auspices of the Office of Basic Energy Sciences, U.S. Department of Energy under Contract DE-AC06-76RLO 1830 with Battelle Memorial Institute, which operates the Pacific Northwest Laboratory.

We thank R. Shepard, R. Littlefield, E. Lusk and R. Stevens for their insight, advice and sharing of Delta experiences, and J.K. Fisher for reading this manuscript. We gratefully acknowledge the excellent work done by the staff of the CCSF without which our work would not have been possible.

References

- [1] P. Saxe, H.F. Schaefer III and N.C. Handy (1981) *Chem. Phys. Lett.* 79:202
- [2] P.E.M. Siegbahn (1984) *Chem. Phys. Lett* 109:417
- [3] P.J. Knowles and N.C. Handy (1984) *Chem. Phys. Lett.* 111:313
- [4] S. Zarrabian, C.R. Sarma and J. Paldus (1989) *Chem. Phys. Lett.* 155:183
- [5] R.J. Harrison and S. Zarrabian (1989) *Chem. Phys. Lett.* 158:393
- [6] J. Olsen, B.O. Roos, P. Jørgensen and H.J. Aa. Jensen (1988) *J. Chem. Phys.* 89:2185
- [7] R.J. Harrison and N.C. Handy (1983) *Chem. Phys. Lett.* 98:97
- [8] R.J. Harrison and N.C. Handy (1983) *Chem. Phys. Lett.* 95:386
- [9] R.L. Graham, D.L. Yaeger, J. Olsen, P. Jørgensen, R.J. Harrison, S. Zarrabian and R.J. Bartlett (1986) *J. Chem. Phys.* 85:6544
- [10] C.W. Bauschlicher Jr. and S.R. Langhoff (1988) *Theoret. Chim. Acta* 73:43
- [11] C.W. Bauschlicher Jr., S.R. Langhoff, H. Partridge and P.R. Taylor (1986) *J. Chem. Phys* 85:3407
- [12] C.W. Bauschlicher Jr. and P.R. Taylor (1986) *J. Chem. Phys.* 85:2779
- [13] J. Almlöf and P.R. Taylor (1987) *J. Chem. Phys.* 86:4070

- [14] T.H. Dunning Jr. (1989) *J. Chem. Phys.* 90:1007
- [15] J. Olsen, P. Jørgensen, and J. Simons (1990) *Chem. Phys. Lett.* 169:463
- [16] "A *Touchstone DELTA System Description*," Intel Super Computer Systems Division, Intel Corporation (1991).
- [17] i860 and 386 are registered trademarks of Intel Corporation.
- [18] "*Touchstone Delta C system call reference manual*," Intel Corporation, order number 312123-001 (1991).
- [19] R. Littlefield, p. 179 in "*Proceedings of the First Intel Delta Applications Workshop*," edited by T. Mihaly and P. Messina, Caltech, Pasadena CA (1992).
- [20] W. Duch (1990) *Intern. J. Quant. Chem* S24:683
- [21] E.R. Davidson (1975) *J. Chem. Phys.* 17:87
- [22] W. Duch (1985) *Intern. J. Quant. Chem.* 27:59
- [23] This simple and effective trick was suggested to us by E. Lusk (1991).
- [24] R. Littlefield, private communication (1992).
- [25] R. Shepard (1990) *J. Comp. Chem.* 11:45
- [26] M. Aoyagi, R. Shepard and A. Wagner (1991) *Intern. J. Super. Appl.* 5:72
- [27] V. Herrarte and E. Lusk, "*The study of parallel program behavior with UP-SHOT*," Argonne National Laboratory Technical Report 91/15 (1991).

Figure 1: Timings for full-CI calculations on the methyl radical versus the reciprocal of the number of processors used. Times and error bars are as reported and described in Table 3. The annotations in parentheses give the number of processors and the speed-up and efficiency relative to 144 nodes.

Figure 2: Timings for full-CI calculations on the neon atom in the 5s3p2d1f basis versus the reciprocal of the number of processors used. Times and error bars are as reported and described in Table 3. The annotations in parentheses give the number of processors and the speed-up and efficiency relative to 144 nodes.

Figure 3: The number of interrupts generated on each node in servicing the distributed CI vector during one matrix-vector product of the Ne 5s3p2d1f calculation on 144 nodes. The values for servicing the product (σ) vector would be identical. The horizontal line indicates the mean value.

Figure 4: The amount of data sent (in units of double precision floating point words) by each node in servicing the distributed CI vector during one matrix-vector product of the Ne 5s3p2d1f calculation on 144 nodes. The amount of data received in managing the product (σ) vector would be identical. The horizontal line indicates the mean value.

Figure 5: The number of remote requests for CI coefficients made by each node during one matrix-vector product of the Ne 5s3p2d1f calculation on 144 nodes. The values for scattering of the product (σ) vector would be identical. The horizontal line indicates the mean value.

Figure 6: The number of remote CI coefficients (in units of double precision floating point words) gathered by each node during one matrix-vector product of the Ne 5s3p2d1f calculation on 144 nodes. The values for scattering of the product (σ) vector would be identical. The horizontal line indicates the mean value.

Table 1: Dimensions (number of orbitals, electrons, configuration state functions and corresponding determinants) and operation counts associated with the full-CI calculations discussed in the text. The operation count is the measured number of floating-point operations performed by the matrix-multiply in the two-electron interaction.

System	Active orbitals	Active electrons	Symmetry	No. of CSF	No. of dets.	Operation count/ 10^{11}
CH ₃ cc-pVDZ	28	7	C_s	14,966,406	33,540,030	1.51
Ne 5s3p2d1f	30	8	D_{2h}	21,580,965	93,896,477	0.88
Ne 6s4p3d	32	8	D_{2h}	36,835,620	161,650,624	1.74
CH ₄ cc-pVDZ	33	8	C_{2v}	94,930,032	418,639,400	8.96

Table 2: Energies (in Hartree) associated with the full-CI calculations discussed in the text. Note the the methyl radical full-CI was performed with an MCSCF canonical frozen core orbital, while the SCF and SDCI numbers reported here used an SCF canonical orbital.

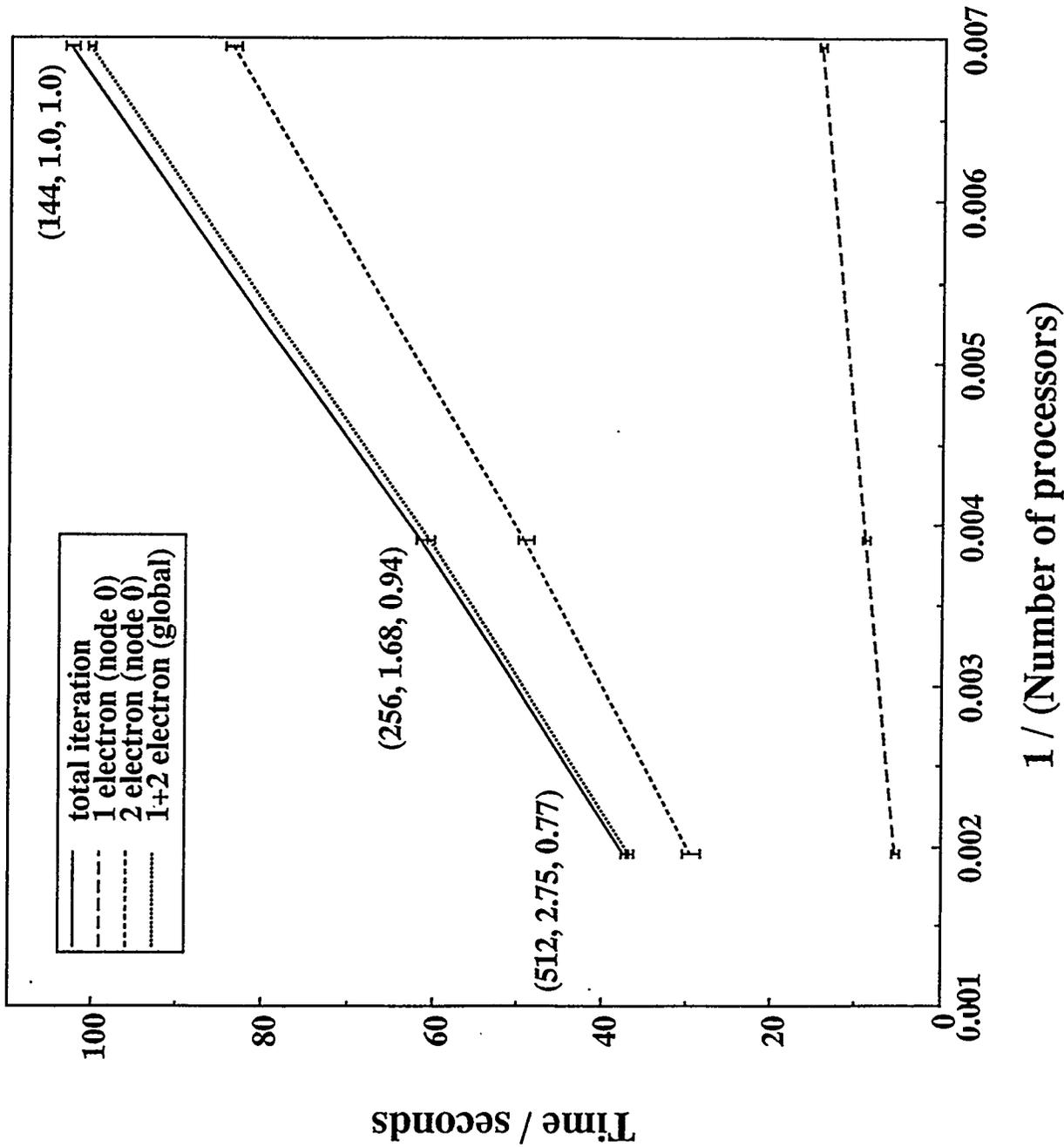
System	SCF	SDCI	Full-CI
CH ₃ cc-pVDZ	-39.558810	-39.706105	-39.714520
Ne 5s3p2d1f	-128.524013	-128.784244	-128.794567
Ne 6s4p3d	-128.524432	-128.717361	-128.725161
CH ₄ cc-pVDZ	-40.198638	-40.375067	-40.387319

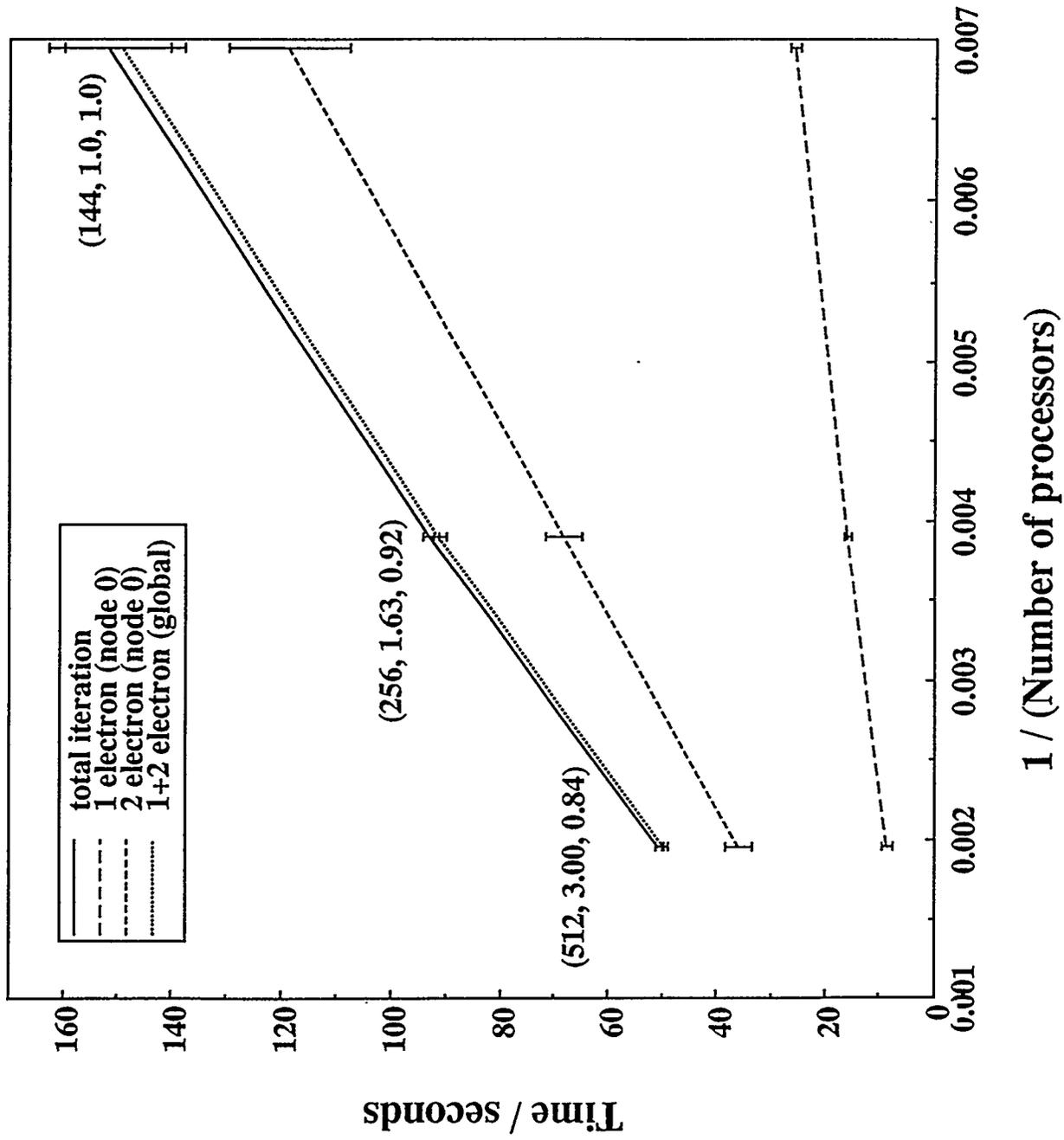
Table 3: Timing and performance statistics of the full-CI calculations discussed in the text. The reported times (seconds) are computed as the arithmetic mean over all matrix-vector products performed, with error bars given as the square root of the variance in the times. The 1-electron and 2-electron times are local times from process (or node) zero. The global time reflects the total (machine wide) time taken to perform a matrix-vector product. The total time is the global time plus any time in the Davidson algorithm, including I/O. Only the CH₄ calculation was unable to buffer all I/O in memory. The GFLOP column reports a lower bound to the computation rate in GFLOP/s computed from the operation count in Table 1 and the global and total times, the latter being in parentheses.

System	# nodes	1-elec.	2-elec	global	total+I/O	GFLOP
CH ₃ cc-pVDZ	144	14.2±0.8	83.6±0.9	100.4±0.4	102.8±0.8	1.50 (1.47)
	256	8.8±0.5	49.2±1.0	60.3±0.5	61.4±0.6	2.50 (2.46)
	512	5.2±0.4	29.5±1.0	36.7±0.4	37.3±0.4	4.11 (4.05)
Ne 5s3p2d1f	144	25.8±1.0	118.7±10.9	149.0±11.2	151.5±11.3	0.59 (0.58)
	256	15.7±0.7	68.2±3.4	91.2±1.1	92.8±1.4	0.96 (0.95)
	512	8.5±1.0	35.9±2.4	49.6±0.5	50.5±0.7	1.77 (1.74)
Ne 6s4p3d	256	24.9±0.9	112.8±2.2	142.1±1.4	144.7±1.8	1.23 (1.20)
	512	14.5±1.0	58.8±2.7	79.4±0.9	81.6±1.2	2.19 (2.13)
CH ₄ cc-pVDZ	512	34.6±1.9	184.5±3.5	226.5±2.4	520.0±223.4	3.96 (1.72)

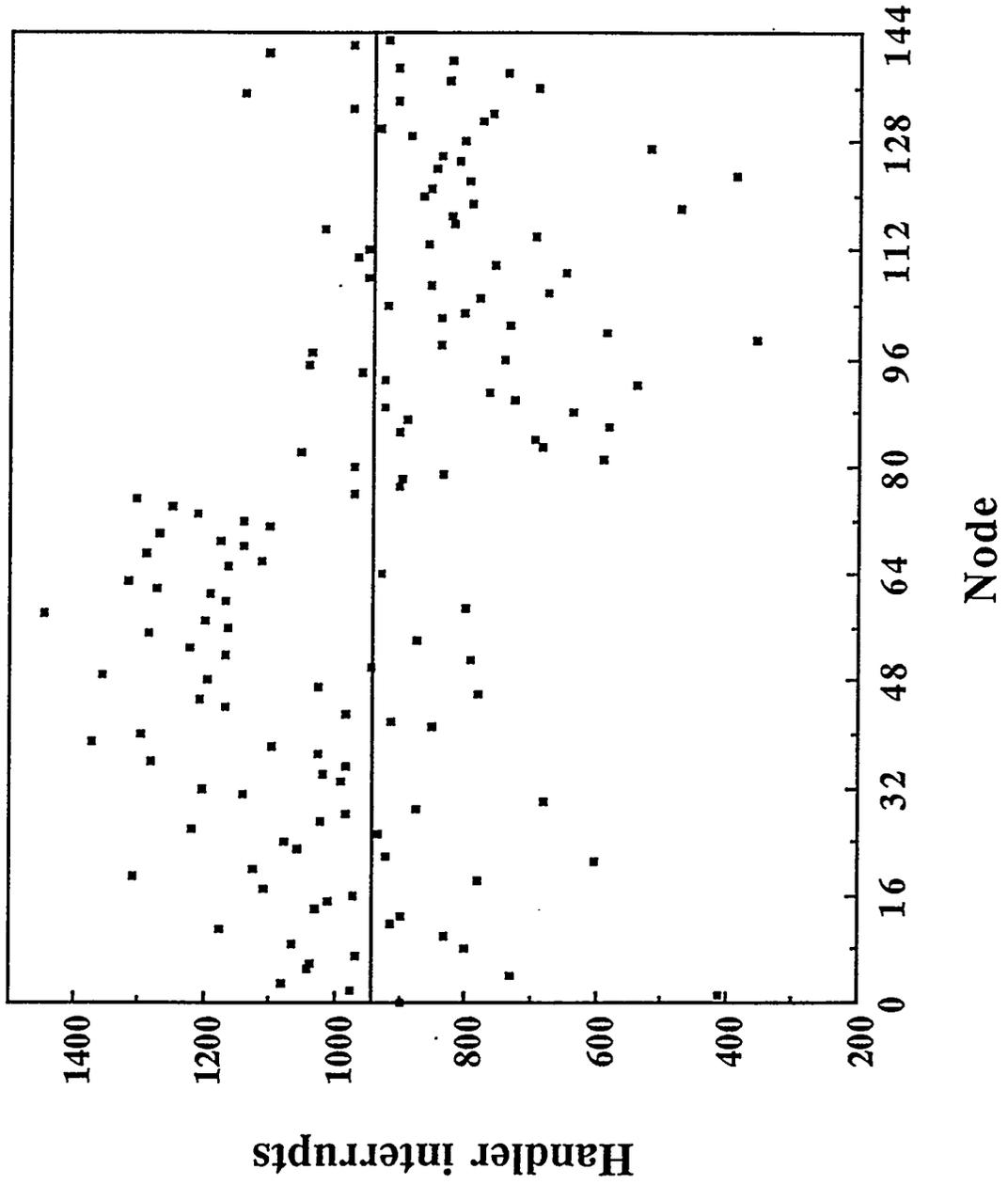
Table 4: Summary of message-passing activity during a single matrix-vector product. The total data sent is the sum of all data sent by handlers for the CI and product vectors. The number of messages sent is computed likewise, but excludes the requests and acknowledgments in the protocol required to bypass normal flow control. The communication rate is computed from the total data and the global times in Table 3, and is thus an average rather than peak communication rate.

System	Number of Nodes	Total Data Sent (Mbyte)	Communication Rate (Mbytes/s)	Messages Sent (thousands)
Ne 5s3p2d1f	144	8037	54	544
	256	7842	86	279
	512	7934	160	405
Ne 6s4p3d	256	12986	91	784
	512	12580	158	527
CH ₄ cc-pVDZ	512	33659	149	4188

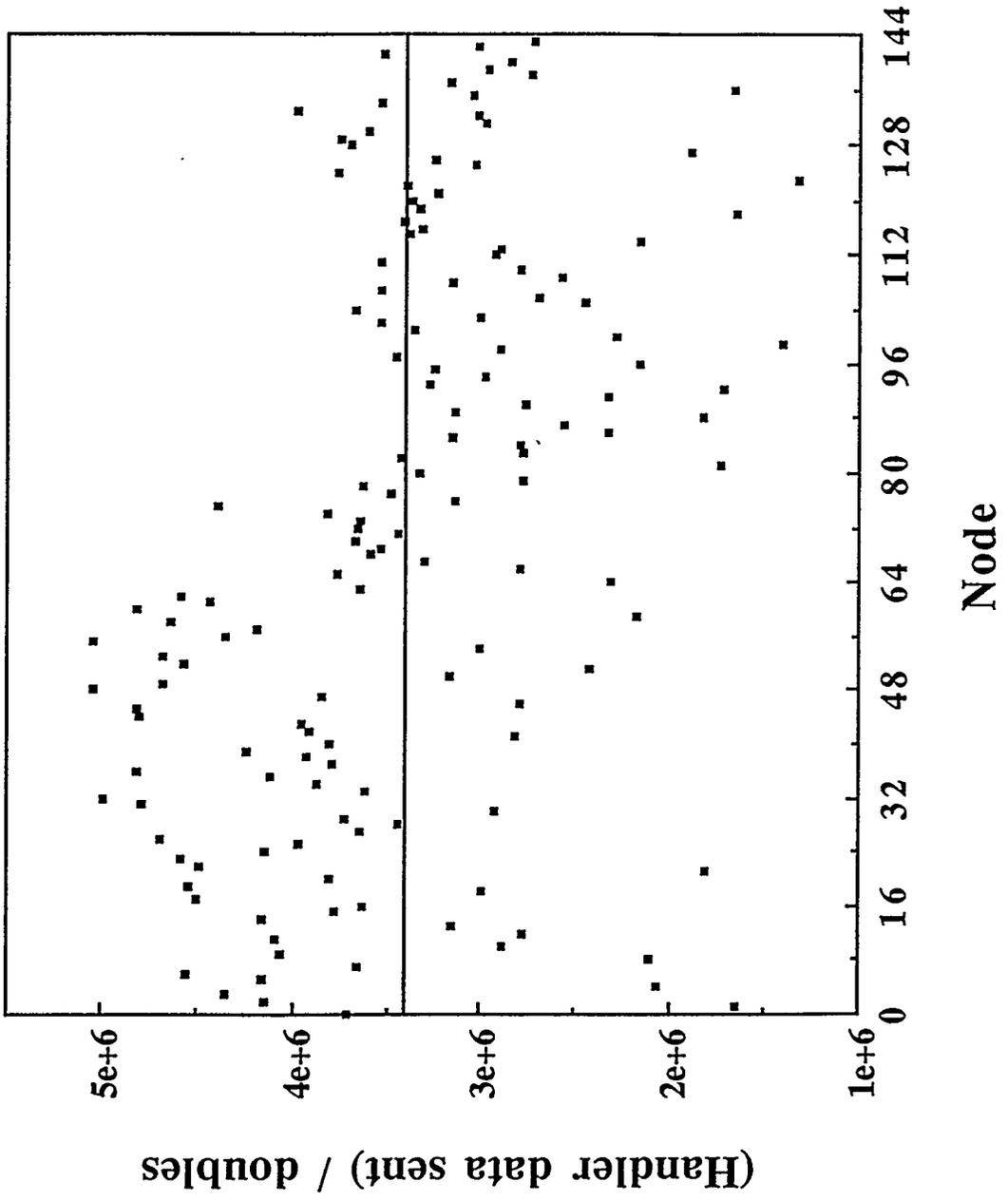




10/10/00



0. 2000



0.0000

