Four-Index Transformation on Distributed-Memory Parallel Computers

Lawrence A. Covick and Kenneth M. Sando

Department of Chemistry, University of Iowa, Iowa City, Iowa 52242

Received 29 August 1989; accepted 7 May 1990

Because it has $\mathcal{O}(N^5)$ operations, a low computation to data transfer ratio, and is a compact piece of code, the four-index transformation is a good test case for parallel algorithm development of electronic structure calculations. We present an algorithm primarily designed for distributed-memory machines. Unlike the previous algorithm of Whiteside et al., ours is not designed with a particular architecture in mind. It is a general algorithm in the sense that not only can it be used on some common architectures but it can utilize some of the advantages inherent in each. In addition, we present formulas predicting that there would be a twofold decrease in communication time if our algorithm was used instead of that of Whiteside et al., on a square array of processors and up to an N-fold decrease if the two algorithms were implemented on a hypercube.

INTRODUCTION

Since the improvement in standard sequential computing speed is limited, the use of several computers concurrently on one problem is an attractive alternative for large problems. One of the larger computational problems in the physical sciences is molecular electronic structure calculation. For large basis sets the four-index transformation has a greater number of operations than most steps in electronic structure calculations, which makes it an obvious candidate for parallel algorithm development. It also has a low computation to data transfer ratio, thereby creating a potential bottleneck for parallel correlation energy calculations.

The four-index transformation is used, almost universally, as a step in the calculation of the correlation energy of a molecule. The need for the four-index transformation arises because it is far easier to carry out correlation energy calculations in a molecular orbital basis, but the two-electron integrals are originally evaluated in an atomic orbital basis. The formula is simple,

$$(ij\,|\,k\ell) = \sum_{\mu}\sum_{\nu}\sum_{\lambda}\sum_{\sigma} C_{\mu i}C_{\nu j}C_{\lambda k}C_{\sigma \ell}(\mu\nu\,|\,\lambda\sigma)\,.$$

Here, $(ij\,|\,k\ell)$ is a two-electron integral in the molecular orbital basis, $(\mu\nu\,|\,\lambda\sigma)$ is an integral in the atomic orbital basis, and the matrix **C** contains the molecular orbital coefficients determined by some type of self-consistent-field calculation. Each sum runs from 1 to N, where N is the number of basis functions. The problem requires a number of floating point operations of $\mathcal{O}(N^5)$.

The permutational symmetry of the integrals

$$(\mu\nu\,|\,\lambda\sigma) = (\nu\mu\,|\,\lambda\sigma) = (\mu\nu\,|\,\sigma\lambda) = (\nu\mu\,|\,\sigma\lambda) = (\lambda\sigma\,|\,\mu\nu) = (\lambda\sigma\,|\,\nu\mu) = (\sigma\lambda\,|\,\mu\nu) = (\sigma\lambda\,|\,\nu\mu)$$

is exploited in optimal algorithms[1–5] for serial, or shared-memory parallel computers. The number of floating-point multiply operations (FPMOs) in optimal algorithms is equal to $25N^5/24$, if the number of molecular orbitals is equal to the number of atomic orbitals. Molecular point-group symmetry has been used to further reduce the number of operations and amount of storage required for the transformation[3] and a parallel implementation as part of an MP2 electronic structure program has been recently accomplished.[5]

The algorithms discussed in this paper are primarily for distributed-memory machines, i.e., ones where each processor has its own local high-speed memory that is not directly addressable from other processors. Interprocessor communication is possible, but over data channels that are usually slow compared to local memory access. The main advantage distributed-memory machines have over those with a shared-memory is an increase in total memory access band-width with increasing number of processors.

In adapting software for distributed-memory computers, a major concern is to minimize the amount of inter-processor communication as compared to the amount of processing done between communications and to take advantage of any possible communication asynchronous with numerical calculation.[6,7] These can also be im-

Journal of Computational Chemistry, Vol. 11, No. 10, 1151–1159 (1990)

CCC 0192-8651/90/101151-09$04.00

portant concerns for shared-memory machines for as the number of processors increases in a concurrent shared-memory machine, memory access slows due to bank conflicts, limited bandwidth for memory access, and other factors. Therefore, code optimized for a distributed-memory machine may also work well on a shared-memory one.

Whiteside, Binkley, Colvin, and Schaefer[6] have shown that asynchronous communication, available on some distributed-memory machines, may be used to give nearly linear speed-ups, i.e., ones directly proportional to the number of processors used. In discussing this algorithm (and our algorithm which follows) we will distinguish between "processor" and "task" and we will use the notation of Whiteside et al. A "processor" is a computer while a "task" is an algorithmic construct roughly equivalent to a job for a theoretical processor. The symbol $(\mu\nu\,|\,\lambda\tilde{\sigma})$ represents a set of integrals with fixed values of μ and ν and all values of λ and σ. Greek letters are used to refer to atomic orbitals and Roman letters to molecule orbitals.

The algorithm of Whiteside et al., utilizes N^2 tasks, with coordinates (μ, ν), which can be distributed over as many as N^2 processors. The system is initialized so that task (μ, ν) has access to the $N(N + 1)/2$ integrals of the symmetric array $(\mu\nu\,|\,\tilde{\lambda}\tilde{\sigma})$, and the molecular orbital coefficient array **C**. The transformation over the first two indices, from $(\mu\nu\,|\,\tilde{\lambda}\tilde{\sigma})$ to $\mu\nu\,|\,\tilde{k}\tilde{l})$, is performed locally, but in order for the third and fourth indices to be transformed, it is necessary to exchange integrals among the tasks. Since the mechanics of the third and fourth index transformations are similar we will only discuss the former.

Whiteside et al. assume both the processors and mapped tasks are arranged in a square array (each row of tasks is distributed along a row of processors) with communication along a row or column of processors (tasks). The tasks pass the triangular arrays $(\mu\nu\,|\,\tilde{k}\tilde{l})$ along a row asynchronously while computing $(\mu j\,|\,\tilde{k}\tilde{l})$ and relabelling themselves (μ, j). Each task must receive $N(N + 1)/2$ integrals from each of the other $N - 1$ tasks in its row.

The use of communications asynchronous with numerical calculations permits complete overlap of calculations with communications (producing ideal speed-ups) if $t_{\text{comp}} \geq t_{I/O}$. The number of basis functions necessary to attain this depends upon the relative MFLOPs and data communications rates of the machine, the number of processors, and the algorithm. Whiteside et al. find nearly ideal speed-up for 16 basis functions on a 32 node Intel hypercube with slow floating-point processors (with faster processors communications overlap, and hence ideal speed-up, is more difficult to attain). This speed-up is gained at the expense of more FPMO's ($5N^5/2$ are needed), because they are unable to use the full permutational symmetry of the molecular integrals in their algorithm.[8]

In this paper we present a new parallel four-index transformation algorithm. We also present formulas that predict that up to an N-fold decrease in communication time is possible if our algorithm is implemented on a hypercube architecture rather than on a square array of processors.

METHODS

The new algorithm we are presenting is called a minimum symmetry one. Minimum symmetry algorithms do not totally ignore the permutational symmetry of the integral indices, but they do not make use of the full symmetry. The advantage in these algorithms over maximum symmetry ones is that they can readily use communications asynchronous with numerical calculations, while the disadvantage is that they require more total FPMOs than maximum symmetry algorithms. (The algorithm of Whiteside et al. is an example of a minimum symmetry algorithm).

Our algorithm has two advantages over that of Whiteside et al. (1) less information is passed between tasks; and (2) the final integral distribution is such that "super-integrals" can be formed locally at each process. The initial distribution and the transformation of the first two indices is similar in the two algorithms. The difference between our algorithm and that of Whiteside et al. is in the transformation of the last two indices.

After transformation over the first two indices, task (μ, ν) contains the array $(\mu\nu\,|\,\tilde{k}\tilde{\ell})$. To carry out the third index transformation, rows of the array $(\mu\nu\,|\,\tilde{k}\tilde{\ell})$ are passed along a row of tasks while computing $(\mu\tilde{j}\,|\,k\tilde{\ell})$ at each task, which is now labelled (μ, k). Because it is confusing to follow the message passing, we will illustrate by considering what happens at task (1,1). Task (1,1) keeps the row $(11\,|\,1\tilde{\ell})$, sends the row $(11\,|\,2\tilde{\ell})$ to task $(1,2),\ldots$ and sends the row $(11\,|\,N\tilde{\ell})$ to task $(1,N)$. In the meantime, task $(1,1)$ accumulates the elements $(1\tilde{\nu}\,|\,1\tilde{\ell})$. After the message passing has completed or, if asynchronous communications is used, while it is taking place, task $(1,1)$ performs the transformation of $(1\tilde{\nu}\,|\,1\tilde{\ell})$ to $(1\tilde{j}\,|\,1\tilde{\ell})$. A schematic algorithm, in the spirit of Figure 5 of Whiteside et al. is shown in Figure 1. Here, for simplicity, we assume that array $(\mu\nu\,|\,\tilde{k}\tilde{\ell})$ has been filled out to be a square array. Also, for simplicity, we assume a complete graph architecture (each processor directly connected to all other processors) with two-way synchronous

```
subroutine trans(xints,yints,bufr,bufs,c,mu,nu,nb)
  comment: Perform the transformation on the index ν.  xints holds the
      integrals in the A.O. basis and will hold the integrals in the
      M.O. basis, yints is a temporary array that will hold the
      integrals received from other tasks.
      bufr amd bufs are scratch arrays big enough to
      hold one row of xints, c is the M.O. coefficient matrix,
      (mu,nu) are the coordinates of this process, and nb is the
      number of basis functions.
    dimension xints(nb,nb),yints(nb,nb),c(nb,nb),bufr(nb),bufs(nb)
    comment:  initial nu-th row of yints
    do ℓ=1,nb
      yints(nu,ℓ)=xints(nu,ℓ)
    enddo
    comment: exchange integrals
    nr=nu
    ns=nu
    do n=1,nb-1
      nr=nr-1
      if(nr.eq.0)nr=nb
      ns=ns+1
      if(ns.gt.nb)ns=1
        do ℓ=1,nb
          bufs(ℓ)=xints(ns,ℓ)
        enddo
      sendrow(bufs,ns)
      recvrow(bufr,nr)
        do ℓ=1,nb
          yints(nr,ℓ)=bufr(ℓ)
        enddo
    enddo
    comment: Transform over ν
    do ℓ=1 to nb
        do j=1 to nb
          xints(j,ℓ)=0.0
            do ν=1 to nb
              xints(j,ℓ)=xints(j,ℓ)+c(ν,j)*yints(ν,ℓ)
            enddo
         enddo
    enddo
    return
end
```

Figure 1. Schematic code for the transformation over the third index. Several simplifying assumptions have been made for clarity.

communications between processors. The subroutine sendrow(buf, n) is defined to send the contents of buf to task n in the same row, and the subroutine recvrow(buf, n) is defined to receive into buf from task n in the same row. This algorithm is included only to illustrate the logic of the third index transformation. A practical algorithm will include asynchronous communication, pipelined messages, and will not be restricted to a complete graph architecture.

The transformation over the fourth index is performed similarly, except that integrals are sent along columns of tasks rather than rows. The final result is that task (j,k) contains the integrals $(\tilde{\imath}j \,|\, k\tilde{\ell})$. One advantage of this algorithm over that of Whiteside, et al. is that the information needed to form "super-integrals" often used in post-SCF procedures[1] is available at each task

$$(jk \,\|\, i\ell) = (ij \,|\, k\ell) - (\ell j \,|\, ki).$$

Another advantage is in the amount of communication required. The total information each task requires from other tasks is $\mathcal{O}(N^3)$ in the algorithm of Whiteside, whereas in our algorithm it is $\mathcal{O}(N^2)$. If this savings in data requirements can be translated into a savings in communication time, the efficiency will be greater, especially when a large number of processors is used. Unfortunately, because some messages must be sent to distant processors the full savings can only be guaranteed for machines with complete graph architecture; however, we can use the pipelining of messages in square arrays of tasks and the special attributes of the hypercube architecture for point-to-point communications in order to take full advantage of this algorithm.

In pipelining, each message contains more than one unit of data (array row or column). Pipelining is accomplished by having a processor send to its nearest neighbor not only its data for that neighbor, but also its data for processors beyond. In addition, it passes on data received from other processors that are intended either for its nearest neighbor or for processors more distant. This procedure is illustrated for the case of data communication required for the third index transformation in Figure 2. We assume a square array of processors with communication along a row and the row connected into a ring. Because

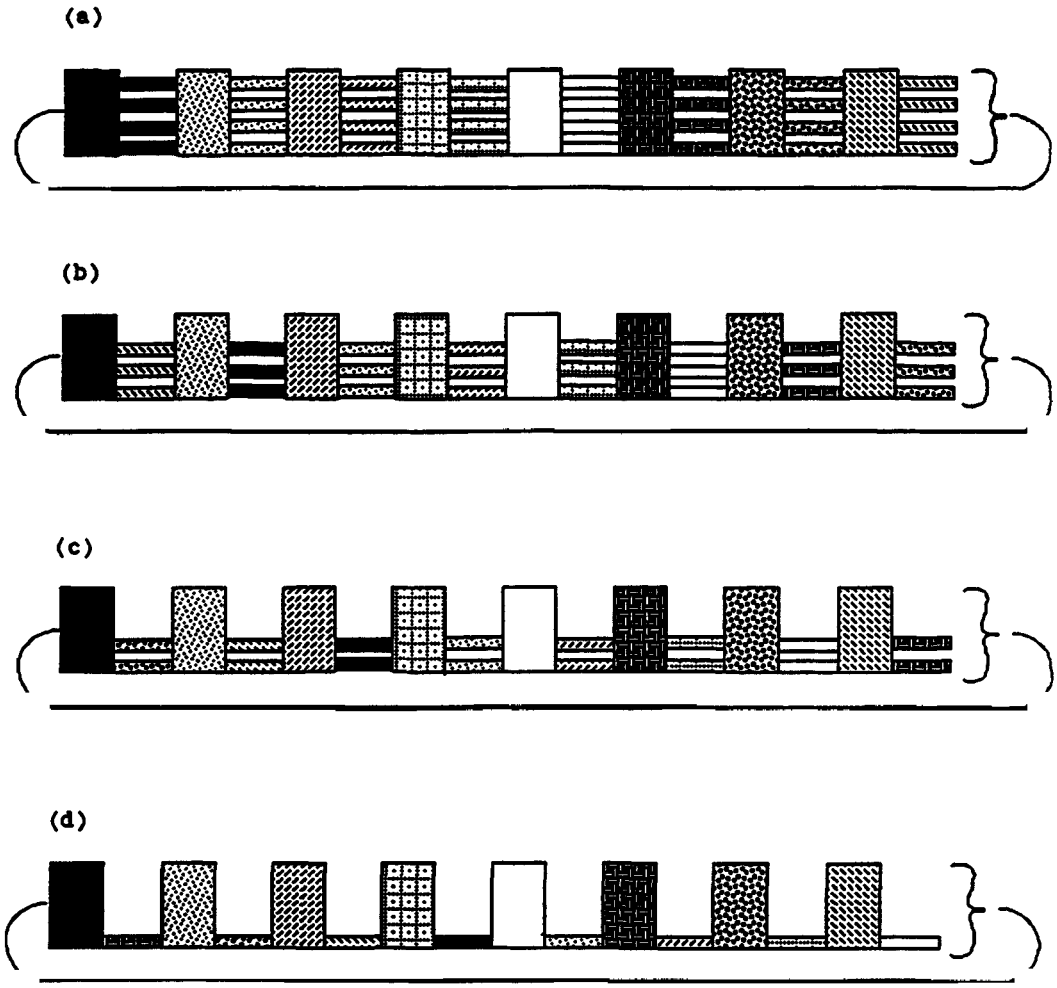

Figure 2. An example of pipelining messages. Each processor sends a message (whose pattern in the above diagram matches the pattern of its processor of origin) containing data for the most distant processor and also for all of the processors in between (a). The receiving processors then remove the portion of the message intended for them and then relay the rest of the message to more distant processors ((b) and (c)) until the most distant processor receives its portion of the original message (d). This procedure is followed by a similar one in the reverse direction to complete the message passing.

communication can occur clockwise or counterclockwise around the ring, the most distant processor is at a distance of $P^{1/2}/2$, where P is the number of processors.

A final point relates to the number of FPMOs required. Since this algorithm only uses one of the permutational symmetries for the transformation of only one of the indices the number of FPMOs is $7N^5/2$ for this algorithm. This can be compared to a value of $25N^5/24$ for a maximum symmetry algorithm. In concurrent processing, this penalty of approximately a factor of 3.5 in FPMOs vanishes if the number of processors that can be effectively utilized goes up by a factor of 4 or more.

RESULTS

The following formulas are estimates of the communication time for the transformation of either the third or the fourth index. Both times should be identical and the total communication time should be proportional to twice the time for either step. It should be emphasized that these are estimates based on the amount of data communicated. Machine dependent parameters such as the overhead involved in sending a message, variations in the allowed maximum size of each "packet" of information being sent, and others are not considered here. In the following formulas P is the number of processors, N is the number of basis functions, M is the smallest integer greater than $N/P^{1/2}$, and integer arithmetic is used throughout. Also, when we refer to a "load-balanced" system, we mean a system that is perfectly load-balanced, i.e., $N = MP^{1/2}$.

The communication time depends in a detailed way upon the distribution of tasks among the processors. If one uses a square array of processors, the algorithm of Whiteside et al., and if we assume the systems are load-balanced, then $t_{1/0}$ (in units of words transferred) for either the third or fourth index transformation has the fol-

lowing formula

$$t_{I/O} = \frac{N^4}{2P^{1/2}} - \frac{N^4}{2P} + \frac{N^3}{2P^{1/2}} - \frac{N^3}{2P} \tag{1}$$

while the load-balanced formula for our algorithm is

$$t_{I//O} = \frac{N^4}{4P^{1/2}} - \frac{N^4}{2P} \tag{2}$$

If the systems are not load-balanced and for the sake of simplicity if $\text{mod}(N/P^{1/2}) \geq (P^{1/2})/2$ (formulas for $\text{mod}(N/P^{1/2}) \leq (P^{1/2})/2$ have the same dependencies on N and P but are more complex) then eq. (1) becomes

$$t_{I/O} = M^2 \frac{N(N+1)}{2}(P^{1/2} - 1) \tag{3}$$

and eq. (2) becomes

$$t_{I/O} = \frac{N}{2}\left(\frac{P}{2} - P^{1/2}\right)M^3. \tag{4}$$

Again for the sake of simplicity, it is assumed that P is an odd number of processors in both eq. (3) and eq. (4).

A direct comparison of eq. (1) and eq. (2) shows that our algorithm has a reduction in the communications time of a factor of 2 for large P. It should be noted that the computation time t_{comp} is $\mathbb{O}(N^5 * P^{-1})$, and therefore decreases more rapidly than $t_{I/O}$ as the number of processors increases. If $P = N^2$ (one task per processor), the communication time and the numerical calculation time are both $\mathbb{O}(N^3)$ which means that communication time could dominate depending on the individual machine's communication and computation parameters.

If our algorithm is implemented on a hypercube then it must be known over how many channels each node can communicate simultaneously. We have looked at the extremes where each node can only pass messages over one channel at a time and where the nodes can communicate over all channels simultaneously. We have named the former a "sequentially communicating hypercube," or SCH, and the latter a "parallel communicating hypercube," or PCH.

The communication time depends upon the way in which the tasks are mapped onto the hypercube. We assume a hypercube of dimension $2n$ ($P = 2^{2n}$) and that rows and columns of tasks are mapped onto processor subcubes of dimension n in a manner similar to their mapping onto rows and columns of processors as described above for a square array.

If our algorithm is implemented on a SCH the load-balanced expression is

$$t_{I/O} = n\frac{N^4}{2P} - \log_2(P)\frac{N^4}{4P} \tag{5}$$

while for the PCH implementation the formula becomes

$$t_{I/O} = \frac{N^4}{P} - \frac{N^4}{P^{3/2}} \tag{6}$$

For large P these two expressions differ by $\mathbb{O}(1/4)\log(P)$. Since one is using $(1/2)\log(P)$ times as many data channels in the transformation of each index (remember there are P of these occurring simultaneously) this is reasonable (the extra factor of 1/2 is because not all of the channels are used on all of the messages and this is an average).

For the algorithm of Whiteside et al., a load-balanced implementation on a SCH produces the following formula

$$t_{I/O} = \frac{N^4}{2P^{1/2}} - \frac{N^4}{2P} + \frac{N^3}{2P^{1/2}} - \frac{N^3}{2P} \tag{7}$$

and if the implementation is on a PCH then

$$t_{I/O} = \frac{N^4}{4P^{1/2}} + \frac{N^3}{4P^{1/2}}, \tag{8}$$

when load-balanced. It is interesting to note that eq. (7) is exactly the same as eq. (1), meaning that one would not expect to see any difference in communication time for their algorithm implemented on either a square array or a SCH architecture. It is also interesting to observe that while eq. (8) is roughly 1/2 the $t_{I/O}$ predicted for either the SCH or the square array, it still has the same dependence on P as both of those; consequently, $t_{I/O}$ still decreases at a slower rate than t_{COMP} as P increases. This is not the case for our algorithm.

As one would expect from the above observation for the algorithm of Whiteside et al., the formula for a non-load-balanced SCH implementation reduces to that for a non-load-balanced square array of processors (eq. (3))

$$\begin{aligned} t_{I/O} &= \left(\frac{N(N+1)}{2}\right)M^2 \sum_{i=0}^{n-1} 2^i \\ &= \left(\frac{N(N+1)}{2}\right)M^2(P^{1/2} - 1) \end{aligned} \tag{9}$$

if one makes the substitution $2^n = P^{1/2}$. The corresponding formula for a PCH is

$$t_{I/O} = \frac{N(N+1)M^2P^{1/2}}{4} \tag{10}$$

For our algorithm, a non-load-balanced implementation on a SCH has the following equation for $t_{I/O}$

$$t_{I/O} = \frac{n}{2}NM^3P^{1/2} \tag{11}$$

while that for the identical situation on a PCH is

$$t_{I/O} = NM^3(P^{1/2} - 1) \tag{12}$$

Unlike the results for the square array, the formulas for our algorithm on either type of hypercube produce communication times that decrease at about the same rate as the computation time; consequently, $t_{I/O}$ and t_{comp} should never be proportional to the same order of N, no matter what the value of P is.

Of course, $t_{I/O}$ is only part of the problem. How important it is depends on its magnitude relative to t_{comp}. If t_{comp} is expressed in FPMOs (see Introduction) and none of the integral symmetry is used then $t_{comp} = 4N^5$. Our algorithm uses the permutation symmetry in the transformation of one of the indices while the algorithm of Whiteside et al. uses it in the transformation of three indices; consequently, t_{comp} for our algorithm is 7/2 N^5 FPMOs while that of Whiteside et al. is 5/2 N^5 FPMOs.

Since there is a tradeoff of extra computation time for a savings of communication time with the algorithm we have just presented, a question naturally arises regarding the "crossover" point for these algorithms, i.e., what size machine does one need to use so that our algorithm will take less time for a given problem size than the algorithm presented by Whiteside et al. The answer to this depends on several factors. First, do we want to include the time necessary to arrange the integrals so that local formation of the "super-integrals" is possible? If we want to do this, then how do we accomplish this distribution? Also, we need to know the computational speed of each node, the rate of communication between nodes, the architecture of the system, and whether or not the communication can be overlapped with the computation.

If we assume no overlap of communication with computation, then we can use the following formula[3] to calculate the "effective number of mega floating point operations per second" or EMFLOPs

$$\text{EMFLOPs} = \frac{(\text{MFLOPs})\,(\text{EFF})\,(t_{comp})}{t_{comp} + \dfrac{8(\text{MFLOPs})}{TR}(t_{I/O})} \tag{13}$$

where TR is the data transfer rate in megabytes per second, MFLOPs is the number of mega floating point operations achieved by a processor for a vector-scalar-multiply-and-add construct (VSMA) of order six (as defined by Cisneros[9] et al., and used by Bunge[3] et al.), $t_{I/O}$ and t_{comp} are used as above, and EFF is the number of processors multiplied by an efficiency that depends on the load-balancing of the problem on a particular architecture (load-balancing is a problem that we intend to deal with in a future work and EFF drops out of the following crossover time calculations, so we will not discuss it further here). From eq. (13) it can be seen that as the speed of the processors becomes very large relative to the rate of data transfer between processors, the $t_{I/O}$ term will dominate the denominator, but if the converse is true t_{comp} will dominate.

For systems that can completely overlap communication with computation eq. (13) becomes

$$\text{EMFLOPs} = \frac{(\text{MFLOPs})\,(\text{EFF})\,(t_{comp})}{t_{comp} + \left(8\dfrac{\text{MFLOPs}}{TR}t_{I/O} - t_{comp}\right)} \tag{14}$$

If the part of the denominator in parentheses is less than 0 then it can be dropped and only the t_{comp} term will be in the denominator, while if it is greater than 0 the two t_{comp} terms in the denominator will drop out and only the $t_{I/O}$ term will remain.

The calculation of the actual "crossover point" is then obtained by using the following formula:

$$t_{total} = \frac{2\ \text{MFPMOs}}{\text{EMFLOPs}} \tag{15}$$

where MFPMOs is simply mega FPMOs (see Introduction) and the factor of two is used because there is an addition for each multiplication when doing the four-index transformation. The formulas for t_{total} for each algorithm are then set equal to each other and the resulting equation is then either solved analytically or numerically for P.

For all of the following results we have included the time necessary to arrive at the final distribution of integrals needed for "super-integral" formation. For the algorithm of Whiteside et al., we have chosen to calculate timings by combining their algorithm as presented[1] with an additional pipelined communication step similar to the communication scheme for the transformation of one index in our algorithm. This is more favorable to their algorithm than is their own suggestion, which would require $4N^5$ FPMOs and twice the $t_{I/O}$ of their normal algorithm.

We have only used one of the formulas for the Whiteside communication time because in order for the algorithm to efficiently use a PCH's bandwidth it must have up to N times the memory available to it because of the simultaneous arrival of messages through different channels at each node. For large problem sizes and machines this would be a staggering amount of memory, so we believe that in these cases one would be forced to run as an SCH which has the same $t_{I/O}$ formulas as a square array.

The resulting formulas for the "crossover" times between our algorithm and the one of Whiteside et al., are listed in Table I. The formulas are given for the three architectures dealt with in this section. We have also listed for each architecture results for comparison purposes of different pos-

Table I. Crossover Point Formulas: The minimum number of processors, P, for which the total processing time for the algorithm presented is less than that for the algorithm of Whiteside et al.[1]

System[a]	Formula	Asymptotic value of P[b]
Square Array-W	$P^{1/2} = \left(1 + \frac{4}{3N}\right)^{-1}\left(\frac{7(N+4)}{24} + \frac{4}{3N}\right)$	$\frac{N^2}{12}$ (62)
Square Array-B	$P^{1/2} = \left(10 + \frac{8}{N}\right)^{-1}\left(\frac{7N}{4} + 15 + \frac{8}{N}\right)$	$\frac{N^2}{33}$ (160)
Square Array-N	$P^{1/2} = \left(12 + \frac{16}{N}\right)^{-1}\left(N + 8 + \frac{16}{N}\right)$	$\frac{N^2}{144}$ (380)
SCH-W	$2P^{1/2}\left(1 - \frac{1}{N}\right) - \log_2(P^{1/2}) - \frac{7N}{16} - 2 = 0$	$\frac{N^2}{21}$ (380)
SCH-B	$2(P^{1/2} - 1)\left(1 + \frac{1}{N}\right) + \log_2(P^{1/2}) - \frac{7N}{16} = 0$	$\frac{N^2}{21}$ (17)
SCH-N	$2(P^{1/2} - 1)\left(1 + \frac{1}{N}\right) - \log_2(P^{1/2}) - \frac{N}{8} = 0$	$\frac{N^2}{256}$ (500)
PCH-W	$P^{1/2}\left(1 + \frac{1}{P}\right) + \frac{1}{N}(P^{1/2} - 1) - \frac{7N}{32} - 2 = 0$	$\frac{N^2}{21}$ (150)
PCH-B	$P^{1/2}\left(1 - \frac{1}{P}\right) + \frac{1}{N}(P^{1/2} - 1) - \frac{7N}{32} = 0$	$\frac{N^2}{21}$ (14)
PCH-N	$P^{1/2}\left(1 + \frac{1}{N}\right) + P^{1/2}\left(2 - \frac{1}{N}\right) - \frac{N}{16}\left(1 + \frac{16}{N^2}\right) - 32 = 0$	$\frac{N^2}{256}$ (530)

[a]W-only the algorithm of Whiteside et al., overlaps communication with computation, B-both algorithms do, N-neither algorithm does.

[b]All values are rounded off to the nearest whole number and the smallest value of N that gives P to within 10% of the complete formula is in parenthesis.

sible combinations of communication overlap with computation for the two algorithms. In a separate column we have also included the asymptotic values of the formulas and (in parentheses) the value of N for which these give a result to within 10% of the complete formula.

For the formulas given in Table I we have used MFLOPs/TR = 4. It should be emphasized that the formulas are dependent on (MFLOPs/TR)$^{-2}$. If both algorithms can have their communication overlapped with their computation and the computational speed of the processors, in MFLOPs, for a SCH is 8 times the communication speed, in MBYTEs/sec, (which is similar to the values used by Bunge et al.[3]) then the asymptotic value of the "crossover" becomes $N^2/83$. By the same token if the computation and communication speeds are equal then the "crossover" becomes $N^2/1.3$.

From our results it appears as though the question of communication overlap with computation for our algorithm is only relevant to the "crossover point" if the system is a square array. The reduction in communication time for our algorithm eliminates its $t_{I/O}$ as a factor in the "crossover point" determination. On the other hand, the question of overlapped communication is very important for all implementations of the algorithm of Whiteside et al. For the systems in Table I, there is a difference of more than an order of magnitude in the "crossover point" results depending on whether or not the algorithm of Whiteside et al., has its communication overlapped with computation.

It is also evident that the main advantages of algorithm would be seen for large machines using a fine-grained mapping. While this is not the way that things are done presently, for very large basis, say N = 1024, it is probable a fine grained mapping will be required, because even with N^2 processors more than 8 megabytes of memory is needed at each node for it's share of the data. To go to a more course grained approach would require even more memory at each node or a disk drive.

As expected the results show a strong dependency on the MFLOP/TR ratio. Since it is difficult to know what this will be in the future we feel that it is best to use an algorithm that will perform well on any machine, no matter how these relative rates turn out.

CONCLUSIONS

While we do not claim that our minimal symmetry algorithm is the best possible use of a hypercube for the four-index transformation, our algorithm can reduce the communication time by up to a factor of N as compared to the algorithm of Whiteside et al., depending on the number of processors used (up to $P = N^2$). Because the processors used in distributed-memory machines are already much faster than those used by Whiteside et al., reduction in the communication time compared to their algorithm is important if one wants to even maintain the degree of communications overlap they had. If one wants to improve on these results then definitely further reductions in $t_{I/O}$ are needed.

Efficient use of memory in a processor array is another important consideration. Because of the simultaneous arrival of several messages at a node necessary to take advantage of the hypercube architecture the algorithm of Whiteside et al., if used on a hypercube, would take up to N times the memory of our algorithm if $P = N^2$. For a square array the situation is different with our algorithm actually taking 4/3 the memory needed for theirs.

The algorithm we have presented is general because not only can it be used on some common architectures, but it can utilize the communication advantages inherent in each. The implementation on a particular architecture only needs modification of the actual message passing commands. Since the total four-index transformation is such a small piece of code and we would only need to modify a portion of this, it is not inconceivable that a few versions for different architectures could be incorporated into the same electronic structure program.

This algorithm is easily transformed into a truly massively parallel one where $P = N^3$ by simply distributing vectors of integrals among the processors. (The operation count would of course increase to $4N^5$ due to the loss of all permutational symmetry when dealing with the straight forward distribution of these integral vectors among processes.) Single integrals would then be transferred among the processes. Normally this would be terribly inefficient since every message has a certain startup overhead and with messages that small $t_{I/O}$ would be dominated by this overhead. For large values of N this problem should be greatly reduced by pipelining, since the average message is going to be a combination of data from different processors, hence it will be significantly longer than one integral.

The decrease in $t_{I/O}$ and the establishment of a distribution of integrals allowing local computation of the "super integrals" that the algorithm we presented produces do come at a slight cost in computation time. Our algorithm will have 7/5 the computation time of that of Whiteside et al.

We have found that a $(\tilde{\mu}\lambda\,|\,\tilde{\nu}\sigma)$ distribution of integrals can be created from other integral distributions done in a small amount of communication time if our data transfer techniques are used, especially if this is done on a hypercube or other machine with a high communication bandwidth. This suggests the pursuit of variations of the above algorithm and different algorithms that utilize more of the permutational symmetry of the integrals, thereby reducing the number of FPMOs and message sizes without regard to the final distribution of integrals. Also, different integral distributions should be tried to avoid the load-balancing problems seen by Whiteside and implementations are needed to see how these ideas affect todays machines. We are currently working on solutions to these problems, including a maximum symmetry algorithm that uses the full permutational symmetry, and a variant where $P \cong N$.

Finally, since there are many architectures that could be used for a parallel distributed-memory computer it is important to have formulas available to estimate the implemented algorithm's communication time without having to actually write, debug, and run the code on a number of different machines. We have developed formulas for a number of special cases. A few of these have been presented here.

One of the most significant results from the formulas is the dependence of the ratio of the communication time ($t_{I/O}$) relative to the numerical computation time (t_{COMP}) on the number of processors (P). If, for a given algorithm, t_{COMP} decreases more rapidly as P increases than $t_{I/O}$ does, the size of the processor array that can be effectively utilized is likely to be limited. For a hypercube implementation of our algorithm $t_{I/O}$ decreases as the same order of P as does t_{COMP}. We therefore feel that this algorithm is a good starting point on which to build future improved algorithms since theoretically it could have approximately linear speedups independent of the size of the machine on which it is running.

Since a distributed-memory approach to algorithms eliminates memory conflicts we feel that it is superior to a shared memory approach because if communications times can be kept from dominating then linear speedups are possible and there is no upper limit on the amount of future computing power that can be applied to this problem.

Note added in proof: There is now a hypercube available from Intel that has a MFLOPs/TR approximately equal to 34 MFLOP/MBYTE. If this value is used in eqs. (13) and (14), then the asymptotic values

of the crossover formulas decrease by more than a factor of 70, meaning our algorithm should be superior on much smaller machines of this type.

The authors would like to thank Robert Harrison of the Theoretical Chemistry Group at Argonne National Laboratory for helpful discussions.

References

1. S.T. Elbert, "Four Index Integral Transformations: An n^4 Problem?" in *Numerical Algorithms in Chemistry: Algebraic Methods,* Report of NRCC Workshop Aug. 9–11, 1978, p. 129, LBL-8158 (1978).
2. V.R. Saunders and J.H. van Lenthe, *Molec. Phys.*, **48,** 923 (1983).
3. C.F. Bunge, A.V. Bunge, G. Cisneros, and J.P. Daudey, *Comput. Chem.,* **12,** 91, 109, and 141 (1988).
4. J.N. Hurley, D.L. Huestis, and W.A. Goddard III, *J. Phys. Chem.,* **92,** 4880 (1988).
5. M. Dupuis and J.D. Watts, *J. Comput. Chem.,* **9,** 158 (1988).
6. R.A. Whiteside, J.S. Binkley, M.E. Colvin, and H.F. Schaefer III, *J. Chem. Phys.,* **86,** 2185 (1987).
7. J.L. Gustafson, G.R. Montry, and R.E. Benner, *SIAM J. Sci. Stat. Comput.,* **9,** 609 (1988).
8. An algorithm for the Alliant FX-4 that does not use one of the permutational symmetries is found in C.W. Bauschlicher, Jr., *Theor. Chim. Acta,* **76,** 187 (1989).
9. G. Cisneros, C.F. Bunge, and C.C.J. Roothaan, *J. Comput. Chem.,* **8,** 618 (1987).