Similarity Searching in Files of Three-Dimensional Chemical Structures. Implementation of Atom Mapping on the Distributed Array Processor DAP-610, the MasPar MP-1104, and the Connection Machine CM-200[†]

David J. Wild and Peter Willett*

Krebs Institute for Biomolecular Research and Department of Information Studies, University of Sheffield, Western Bank, Sheffield S10 2TN, U.K.

Received June 5, 1993[®]

Atom mapping is an approximate graph-matching procedure that has been developed for similarity searching in databases of three-dimensional chemical structures. Although effective in operation, atom mapping is quite time-consuming when implemented on a conventional, serial processor. We describe three different algorithms for its implementation on a massively-parallel array processor, and then demonstrate the efficiencies of these implementations using an Active Memory Technology Distributed Array Processor DAP-610, a MasPar MP-1104, and a Thinking Machines' Connection Machine CM-200. Experiments with a small file of 2000 three-dimensional structures suggest that one algorithm, called distributed atoms, is the most efficient of the three that we have studied, and that the fastest search speeds are given by the MasPar machine.

1. SIMILARITY SEARCHING USING ATOM MAPPING

One of the most important facilities in any chemical information system is *similarity searching*, where a database is scanned to identify those molecules that are structurally most similar to a user-defined query molecule.[1] A similarity search involves calculating a quantitative measure of intermolecular structural similarity between a user-defined *target* molecule and each of the molecules in a database. The database is sorted into order of decreasing similarity with the target, and interesting compounds from the top of this ranking can then be used as the basis for subsequent searches.[2] The primary rational for such a searching mechanism is the *similar property principle* of Johnson and Maggiora,[3] which states that structurally-similar molecules are expected to exhibit similar properties; thus, if a biologically-active molecule is used as the target in a similarity search, then the top-ranked molecules are expected to have a greater probability of exhibiting the same activity than would be the case if they were selected at random. Accordingly, given a single compound that has been shown to be active in a biological test system, similarity searching provides an excellent way of exploring a database to identify additional active compounds.

Similarity searching in databases of two-dimensional (2-D) chemical structures is normally implemented using fragment-based measures of structural similarity.[4] The recent emergence of database systems for three-dimensional (3-D) structures[5,6] has led to interest in the development of comparable facilities for 3-D similarity searching. However, there is much less consensus as to what should form the basis for similarity searching in databases of 3-D structures, although most of the systems that have been described to date have considered either the steric or the electrostatic factors that are the most important determinants of biological activity.[7] Many of the procedures that have been used for measuring molecular similarity are far too time-consuming if a target molecule is to be matched against many thousands of structures in a database (see, e.g., refs 8–12). However, the past few years have seen the publication of several procedures that are sufficiently fast for use in a database-searching environment (see, e.g., refs 13–20). This paper discusses the use of parallel hardware to increase the efficiency of one such fast similarity-searching procedure; specifically we discuss the implementation of the *atom-mapping* method of Pepperrell *et al.*[14,21,22] on three different types of massively-parallel array processor.[23–26]

Atom mapping provides a global measure of the structural similarity between two 3-D molecules, e.g., a target molecule and a molecule from a database of 3-D structures. The method contains two main stages.[21] In the first stage, the geometric environment of each atom (as represented by the set of interatomic distances between itself and each of the other atoms in the molecule) in a target molecule, T, is compared with the geometric environment of each atom in a database molecule, D. This comparison is used to calculate the interatomic similarity between each of the $N(T)$ atoms in T and each of the $N(D)$ atoms in D. The second stage involves using these interatomic similarities to make matches between atoms in T and atoms in D that lie at the center of comparable environments. The mean of the similarities for the pairs of matched atoms then gives the overall intermolecular similarity.

The interatomic similarities are calculated using the *Tanimoto coefficient*.[2] Here, the similarity $SIM(X,Y)$ between two objects X and Y is given by

$$SIM(X,Y) = \frac{NXY}{NX + NY - NXY}$$

where NX and NY are the total number of attributes in X and Y, respectively, and NXY is the number of attributes that the two objects have in common. In the present context, the attributes are the interatomic distances involving each atom in T (or D). Let P and Q be two atoms in D separated by a distance of $DATABASE\_DIST(P,Q)$ and R and S two atoms in T separated by a distance $TARGET\_DIST(R,S)$. These two interatomic distances are common to T and D if $DATABASE\_DIST(P,Q)$ and $TARGET\_DIST(R,S)$ are the same to within a user-defined tolerance (0.5 Å in our experiments) and if the atomic types of the pairs of atoms P

[†] Presented at the Third International Conference on Chemical Structures, Noordwijkerhout, The Netherlands, June 6–10, 1993. A preliminary version of some of the material in this paper was first presented at the Beilstein Institute Workshop on Similarity in Organic Chemistry, May 25–29, 1992.

* To whom all correspondence should be addressed.

[®] Abstract published in *Advance ACS Abstracts*, January 15, 1994.

and Q and R and S are equivalent. Let the number of distances that match in this way for the Ith atom of T and the Jth atom of D be $COMMON(I,J)$; then the similarity between the two atoms, $SIM(I,J)$, is given by

$$SIM(I,J) = \frac{COMMON(I,J)}{N(T) + N(D) - COMMON(I,J)}$$

The elements of $SIM(I,J)$ comprise the *atom-match matrix*, *SIM*. In the second stage of the procedure, this matrix is repeatedly scanned to identify the most-similar atom in D for each atom in T using a greedy algorithm.[21] A list, *MAPPINGS*, is created that contains the similarities for these pairs of mapped atoms, and then the overall intermolecular similarity is given by

$$\frac{1}{N(T)}\sum_{I=1}^{N(T)} MAPPINGS(I)$$

Having outlined the atom-mapping method, section 2 of this paper gives a brief introduction to massively-parallel array processors. Section 3 presents three different parallel algorithms for atom mapping, and section 4 then describes the implementation of these algorithms on the Active Memory Technology Distributed Array Processor DAP-610, the MasPar MP-1104, and the Thinking Machine Connection Machine CM-200. The results of searching experiments using these three machines are given in section 5, and the paper closes with a discussion of these results and a summary of the main conclusions that we have drawn from this study.

2. MASSIVELY-PARALLEL ARRAY PROCESSORS

Massively-parallel array processors are a class of single instruction-stream multiple data-stream (SIMD) computer[23] that are particularly appropriate for performing computations that involve matrices or arrays. They consist of an array of *processing elements*, or PEs, each of which is connected to a central control unit. There are typically several thousands of PEs, in contrast to most multiprocessor systems which contain much smaller numbers of processors. This is possible because each PE is regular and simple, so that several PEs may be implemented on a single chip using VLSI techniques.

Like most SIMD machines, array processors are generally *loosely coupled*, meaning that each PE has its own local memory rather than sharing a global memory. Interprocessor communication is realized using the interconnection network, which may take the form of either a hard-wired topology or a switching unit, or in some instances both. In an SIMD computer, all interprocessor communication is done under the supervision of the control unit with communication between active processors synchronized. The organization of a loosely-coupled SIMD computer is shown in Figure 1.

Instructions and data are sent from a host computer (which is normally known as the *front end*) or input/output device to the central control unit of the SIMD computer (which is normally known as the *back end*). Data are distributed to PEs and are stored in the local memory of each PE. The control unit instructs each PE to perform equivalent computations on data in its memory synchronously with the other PEs; synchronization is inherent in such a model of computation and therefore far simpler than in conventional multiprocessor systems. An important feature of an array processor is the ability to disable individual PEs by setting a flag, called a *mask*, so that the operation performed in the current instruction cycle is executed only by the active PEs.

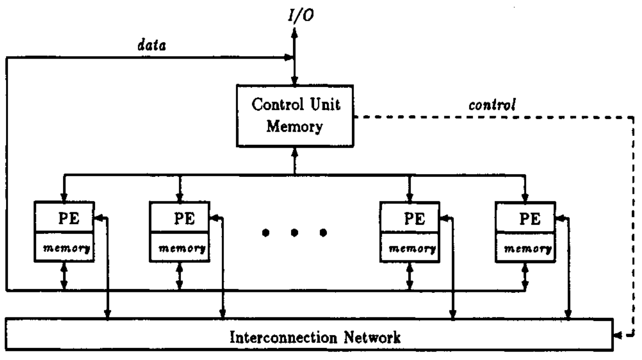

Figure 1. Organization of an SIMD computer (adapted from Hwang and Briggs[23]).

The array-processor architecture is well suited to database applications since it permits a query record to be broadcast from the central control unit for matching against database records that are stored in the array of PEs. We have previously reported several studies of the use of one particular type of array processor, the Distributed Array Processor (DAP),[27] for a range of chemical-database applications,[28,29] and have presented the preliminary results of using this machine for atom mapping.[30] The work reported here involved not only the DAP but also two other types of massively-parallel processors, as discussed below.

The initial experiments used an Active Memory Technology DAP-610 at Queen Mary and Westfield College, University of London (and subsequently at the University of Cambridge). The DAP is an SIMD computer that uses a simple topology in which each PE is connected to its four nearest neighbors.[25] A matrix-based programming model is used, allowing arrays to be declared with two dimensions assigned as parallel, corresponding to the two dimensions of the symmetrical processor matrix. Masking is achieved by the use of Boolean arrays. Our programs used the Fortran Plus language, a version of Fortran with parallel constructs added, with a separate front-end Fortran 77 program to handle I/O. Communication between the front end and the back end is through memory-to-memory transfers over a SCSI or VME bus. Data are loaded into the DAP's PEs by reading it serially into the front-end program and transferring it to the back end. The particular model used here was a DAP-610, with 4096 PEs each with 8 kbytes of local memory connected to a SUN SparcStation 2 by a SCSI bus. Some of the earliest experiments used the same DAP-610 connected to a VAX 11/750 *via* a VME bus.

We then used a MasPar MP-1104 at the Rutherford-Appleton Laboratory, Harwell, U.K. The MP-1 family is based around RISC-processor PEs connected by both a nearest-neighbor mesh (known as the *X-Net*) and a global router, which provides full interconnection *via* a multistage crossbar switch. I/O may be carried out on the array of PEs, although communication may be established with a serial program running on a host machine. Either the host machine or a parallel disk array are used for file storage. Data may be read onto the PEs *en bloc* using a parallel read function. The programming model of this machine is based around the use of parallel variables, which are declared in a similar way to serial variables but which have a separate instantiation on each processor. Operations may then be applied to all, or a subset, of the parallel variables on these processors. Our programs used the MPL language, a derivative of C with parallel extensions. The MasPar MP-1104 used here had 4096 PEs containing 64 kbytes of local memory per PE. It had a 2.9 Gbyte parallel disk array with a transfer rate of 9

```
COMMON(I,J) := 0
X := 1
Y := 1
WHILE NOT End of either list
  IF ABS(TARGET_DIST(I,X)-DATABASE_DIST(J,Y)) ≤ TOLERANCE
  AND Neither already matched
  THEN
    COMMON(I,J) := COMMON(I,J)+1
    X := X+1
    Y := Y+1
  ELSE
    IF TARGET_DIST(I,X)+TOLERANCE < DATABASE_DIST(J,Y)
    THEN X := X+1
    ELSE Y := Y+1
    END IF
  END IF
END WHILE
```

Figure 2. Basic sort–merge algorithm for identifying the number of distances common to two sorted lists.

Mbytes/s and was connected to a DEC 5000 Unix workstation as the front end.

Finally, we have used a Thinking Machines' Connection Machine CM-200 at the University of Edinburgh Parallel Computer Centre, with 16384 PEs containing 32 kbytes of local memory per PE. The programming model of the CM-200 is very similar to that of the MP-1104, involving the use of parallel variables and a language, C*, that is again a derivative of C. Either a standard Unix host machine or a dedicated parallel disk array may be used for file storage: for this work, a parallel disk array known as the *DataVault* was used, which has a peak transfer rate of 25 Mbytes/s.

3. PARALLELIZATION OF THE ATOM-MAPPING METHOD

3.1. Introduction. The calculation of the interatomic similarities is the most time-consuming part of atom mapping. Specifically, the rate-limiting step is the identification of the distances that are common to a pair of atoms (i.e., the *COMMON*(I,J) term in the Tanimoto formula of section 1), and our attempts at parallelization have thus focused on this stage of the method. The basic sequential algorithm[14,21] uses a sort–merge procedure to identify the matching distances. Let *TARGET_DIST*(I,X) be the Xth distance for the Ith target atom, *DATABASE_DIST*(J,Y) be the Yth distance for the Jth database atom with which it is being compared, and *TOLERANCE* be a predefined error tolerance (0.5 Å in our experiments). Then, if the two lists of distances *TARGET_DIST*(I) and *DATABASE_DIST*(J) have each been sorted into monotonically-increasing order, the number of matching distances, *COMMON*(I,J), can be identified using the pseudocode in Figure 2 (where the checking of the atom types during the mapping of the distances has been omitted for the purpose of clarity). This procedure is called $N(T) \times N(D)$ times, once for each pair of atoms; if $N(T)$ and $N(D)$ are approximately the same, call this N, then each such invocation has a complexity of $O(N)$, giving an overall expected time complexity of $O(N^3)$.[21]

In a previous paper,[30] we have discussed three different ways in which this calculation can be implemented on a massively-parallel SIMD machine. In the *distributed distances* approach, the atoms comprising a database molecule are distributed over the PEs, with each PE being assigned one interatomic distance. The similarities are calculated by broadcasting the distances for each query atom in turn, so that the calculation of the Tanimoto calculation is parallelized. The *distributed atoms* approach involves distributing a molecule over the PEs, with each PE being assigned one atom. The similarity is calculated for each query atom in turn, on all the database structure atoms in parallel. Finally, the *distributed molecules* approach involves each processor being assigned one molecule from the database, so that the similarity calculation is done sequentially, but on many molecules in parallel. As noted above, the sequential algorithm has an expected time complexity of $O(N^3)$ for molecules possessing $O(N)$ atoms. The distributed molecules approach has the same complexity, whereas the distributed distances and distributed atoms approaches both have a complexity of $O(N^2)$.[31]

It is possible to combine more than one level of parallelism. Thus, at the distributed atoms level, molecular parallelism can be exploited by processing the atoms from several database molecules at the same time (provided that there are significantly more PEs available than there are atoms in a database molecule). Similarly, atomic and molecular parallelism may be used in the distributed distances method, with the distances for several (or all) of the atoms in one molecule and those of further molecules being processed in parallel. These multiple levels of parallelism were used in our actual programs; however, for clarity of explanation, the discussion of the three algorithms that follows assumes that only a single level of parallelism is being used in each case.

3.2. Distributed Distances. This method is an example of *inner-loop* or *algorithmic* parallelism and seeks to maximize the efficiency with which the pairs of lists of distances, *TARGET_DIST*(I) and *DATABASE_DIST*(J), are processed to identify the distances in common between a pair of atoms I and J.

We have shown previously[30] that it is difficult to produce an array-processor equivalent of the sequential sort–merge algorithm given in the previous section. Firstly, a simple "one-step" parallel operation cannot be performed as each database distance may need to be tested against more than one target distance. This can be circumvented in one of two ways. One way involves making $N(D)$ copies of the database distances (for a database molecule containing $N(D)$ atoms), each copy shifted by one place; this ensures that any one target distance will be tested against a different database distance in each copy, thus enabling a one-step operation to be performed. Alternatively, it is possible to perform $N(D)$ operations, shifting the database-atom distance list one place each time. Both of these options are computationally expensive: the former due to the time for copying and the severe reduction in parallelization (since only $1/N(D)$ as many atoms can be processed at once), and the latter due to the extra $N(D)$ operations that are necessary. In practice, the latter was chosen as the better method. Secondly, and more importantly, it is not possible to ensure that this parallel algorithm will identify the maximum-possible number of matching distances. This is because more than one distance can be mapped at the same time, thus overriding the strict sequential processing of the lists that forms the basis for the sort–merge algorithm. Accordingly, suboptimal similarities may be calculated[30] (although various heuristics are available that serve to minimize the number of times that this occurs[31]).

The algorithm for the distributed distances method can be summarized in the pseudocode of Figure 3 (where "*" denotes all of the elements of one dimension of an array in a parallel section of code).

3.3. Distributed Atoms Method. The distributed atoms method is another example of inner-loop parallelism and involves allocating all of the interatomic distances for a particular database-structure atom to a single PE, so that the distances for a specific query atom can be matched against

```
FOR I := 1 TO N(T)
  FOR J := 1 TO N(D)
    PARALLEL
      Compare the row in the target distance matrix with the
        corresponding row in the database distance matrix, and
        cumulate the numbers of matching distances in COMMON(I,*)
    END PARALLEL
    Rotate the list of distances for the database atoms by one place
  END FOR
  PARALLEL FOR EACH database atom
    Calculate the similarity with the I-th target atom, i.e.,
      SIM(I,*) := COMMON(I,*)/(N(T)+N(D)-COMMON(I,*))
  END PARALLEL FOR EACH
END FOR
```

Figure 3. Distributed distances algorithm.

```
FOR I := 1 TO N(T)
  PARALLEL FOR EACH database atom
    Execute the normal sort-merge algorithm to calculate
      the number of matching distances, COMMON(I,*)
    SIM(I,*) := COMMON(I,*)/(N(T)+N(D)-COMMON(I,*))
  END PARALLEL FOR EACH
END FOR
```

Figure 4. Distributed atoms algorithm.

```
TOTSIM := 0
WHILE MAX(SIM(*,*)) > 0
  TOTSIM := TOTSIM + MAX(SIM(*,*))
  R := Row where maximum value found
  C := Column where maximum value found
  SIM(R,*) := 0
  SIM(*,C) := 0
END WHILE
```

Figure 5. Parallel implementation of the mapping stage.

the distances for all of the database atoms in parallel. This method does not involve parallelization of the distance matching, and an adaptation of the serial merge-type algorithm can thus be used. A pseudocode description of the algorithm is given in Figure 4.

3.4. Distributed Molecules Method. The distributed molecules method is simpler, since it is an example of outer-loop parallelism in which the basic serial algorithm is executed on many molecules in parallel. One molecule is allocated to each PE, and the distance mapping is then carried out for each target atom on each database atom in turn of every molecule in parallel.

3.5. Parallelization of the Mapping Stage. The calculation of the interatomic similarities is the most time-consuming part of atom mapping, and hence the most appropriate for parallelization: it is, however, also possible to increase the speed of the mapping stage. The serial version of this stage uses a shell-sort algorithm[32] to sort the atom-match matrix so that the largest interatomic similarities can be extracted. SIMD machines provide facilities that permit the speed of this stage to be increased. For example, functions exist on the DAP to extract the highest value in a matrix, and these were used to identify the largest similarities in the *SIM* matrix directly, thus enabling the omission of the sorting stage.

In more detail, the DAP algorithm involves locating the highest interatomic similarities in *SIM* for each target atom and then adding these similarities to a total. The two atoms which gave this similarity are then removed from further consideration by setting to zero all of the similarities in the row and column that contained the maximum similarity. This process is repeated until all of the elements of the matrix are zero-valued. The total molecular similarity is then calculated as described previously. This sequence of operations may be written in pseudocode, as shown in Figure 5, where *MAX* is a function which yields the highest value of any matrix that is passed to it. Once *TOTSIM* has been calculated in this way, the global similarity is then simply *TOTSIM*/*N*(*T*).

An analogous, but more efficient, procedure was used on the other two machines (as described in sections 4.2.2 and 4.3.1).

4. IMPLEMENTATION DETAILS

4.1. Implementation on the DAP-610. *4.1.1. Distributed Distances.* The distances are arranged on the DAP processor matrix so that each row of PEs contains the list of distances for one atom. This is done using a two-dimensional array, *DATABASE_DIST*, where *DATABASE_DIST*(*I*,*J*) contains the *J*th distance for the *I*th atom. Matching is done by storing the target distances for a specific target atom in a vector and then replicating this vector to produce a matrix that can be compared with the database-molecule distances. After each comparison, the array *DATABASE_DIST* is rotated (as described in section 3.2 above) so that each database distance is compared with each target distance.

During this first stage, a Boolean array is maintained to indicate which of the database distances in the PEs were successfully matched to target distances. The number of distances in common for each atom is easily obtained by performing a summation of each row of PEs in this Boolean array (which can be done in parallel), thus permitting the calculation of the similarity between each database atom and the current target atom. As these similarities are obtained, they are placed into a parallel array *SIM*(*I*,*J*) in which the *IJ*th element contains the similarity between the *I*th target atom and the *J*th atom of the database molecule (i.e., that currently stored in the PEs).

Target and database atoms may then be mapped using Fortran Plus functions that enable the highest value in a matrix to be identified, as detailed in section 3.5 above. Similar procedures were used for this stage in the other algorithms and machines.

The 4096 PEs in a DAP-610 are in a 64 × 64 array, and this means that the data for 64 (non-hydrogen) atoms can be processed in parallel; the molecules in the data set used in our experiments were sufficiently small to enable an average of three molecules to be processed together. Although a limit of 64 atoms/molecule is assumed, it is easy for the algorithm to be adapted for a greater number of atoms, as the DAP allows the implicit use of virtual processors, with one PE acting as several virtual PEs. Similar comments apply to the MP-1104 and the CM-200.

4.1.2. Distributed Atoms. In the distributed atoms method, sections of the database are loaded into the processor array such that all of the distances for one atom reside on one PE. This is achieved on the DAP using a three-dimensional array, the *IJK*th element of which contains the *K*th distance of the atom on the *IJ*th PE.

The sort–merge algorithm is suitable for use in the distributed atoms method, but the programming model of the DAP renders such an implementation infeasible, since it would be necessary to use matrices to keep track of the current "positions" in the target-atom and database-atom distance lists and to keep count of the number of mappings made. However, this form of indexing is not supported on the DAP, a limitation which has been noted in previous studies of the use of the DAP for database-processing applications.[33] Instead, an exhaustive comparison had to be implemented, working on sorted lists to ensure that the maximal numbers of distances were matched. As this method involves comparing every distance of the target atom with every distance of the database-structure atom, it requires $O(N(T) \times N(D))$ (i.e., $O(N^2)$) time to execute, rather than $O(N(T) + N(D))$, i.e.,

```
FOR I := 1 TO N(T)
  FOR J := 1 TO N(T)
    FOR K := 1 TO N(D)
      PARALLEL FOR EACH database atom
        Compare the J-th distance of the I-th target atom, TARGET_DIST(I,J),
          with the K-th database distances, DATABASE_DIST(*,K)
        Cumulate the number of matched distances in COMMON(I,*)
      END PARALLEL FOR EACH
    END FOR
  END FOR
  PARALLEL FOR EACH database atom
    Calculate the similarity with the I-th target atom, i.e.,
      SIM(I,*) := COMMON(I,*)/(N(T)+N(D)-COMMON(I,*))
  END PARALLEL FOR EACH
END FOR
```

Figure 6. DAP implementation of the distributed-atoms algorithm.

$O(N)$, for the sequential algorithm (and this must then, of course, be repeated for each of the $N(T)$ target atoms). The exhaustive comparison may be written in pseudocode, as shown in Figure 6.

4.1.3. Distributed Molecules. It was not possible to implement the distributed molecules method on the DAP, as the local memory needed to hold all of the data for one molecule exceeded the 8 kbytes available on each PE in the DAP-610 that was used for this work.

4.2. Implementation on the MP-1104. *4.2.1. Distributed Distances.* It was decided not to implement the distributed distances method on the MasPar-1104 (and also the CM-200) because of its inherent inaccuracy (as discussed in section 3.2 above) and because of its poor performance in the DAP experiments (as detailed in section 5 below).

4.2.2. Distributed Atoms. The programming model of the DAP meant that we could not implement a parallel version of the serial sort–merge algorithm, this causing us to use the far more inefficient procedure that has been detailed in section 4.1.2. Conversely, the programming model of the MasPar series allows us to implement the serial algorithm with only a few, minor modifications, using parallel indexing and counting variables and using the parallel IFs and WHILEs that are available in the MPL language for the implementation of the various control structures.

The flexible interprocessor communications on the MP-1104 also mean that it is possible to perform the mapping stage more efficiently than was possible using the DAP (as discussed in section 3.5). The target atom that is found to be the most similar to the database atom stored on a specific PE is identified, and this information is passed to those neighboring PEs that deal with atoms from the same molecule. The best matches may then be identified, enabling the intermolecular similarity to be calculated with great efficiency.

4.2.3. Distributed Molecules. The distributed molecules implementation of the similarity-calculation stage has a similar structure to the distributed atoms implementation, in that it uses a modified version of the serial sort–merge algorithm. However, it is necessary to introduce an extra level of complexity, as each PE now contains data for several atoms. Therefore the sort–merge algorithm is applied *MAXATS* times, where *MAXATS* is the number of atoms in the largest database molecule that is to be processed, with one atom on each PE being processed on each iteration. As the data for a whole molecule resides on a single PE, the mapping stage is carried out in a serial manner on all of the PEs in parallel.

4.3. Implementation on the CM-200. *4.3.1. Distributed Atoms.* The similarity of the CM-200 programming model to that of the MasPar MP-1104 meant that the structure of the code for the distributed atoms algorithm on the two machines was essentially the same, with extensive use being made of the parallel variables and constructs available in C*

Table 1. Speedup Based on CPU Times on an ESV-30 Workstation for the DAP-610, the MP-1104, and the Connection Machine CM-200 Using the Distributed Distances (DD), Distributed Atoms (DA), and Distributed Molecules (DM) Approaches to Parallelization[a]

target size	ESV-30 time	DAP-610		MP-1104		CM-200	
(atoms)	(CPU s)	DD	DA	DA	DM	DA	DM
6	0.6	0.2	0.4	0.8	0.6	0.2	0.0
9	1.3	0.3	0.5	1.0	0.8	0.3	0.0
12	8.1	1.2	2.4	4.7	3.4	1.4	0.2
15	22.6	2.7	4.9	9.7	7.0	2.7	0.3
18	31.9	3.2	5.2	10.6	7.2	3.0	0.3
21	72.3	5.8	9.0	19.5	14.1	5.8	0.6
24	147.9	9.7	14.5	30.1	23.4	9.8	0.9
27	150.1	8.1	11.6	27.0	19.0	8.4	0.7
30	191.4	8.7	12.0	30.8	21.8	9.4	0.9
33	205.0	8.0	10.4	27.9	19.8	9.9	0.8

[a] In each case, the speedup listed is the mean of four target molecules of the given size.

on the CM-200. However it was necessary to perform some calls to the CM-200's low-level instruction set, PARIS,[34] in order to obtain maximum performance.

The mapping stage was performed using near-neighbor communications, as described in section 4.2.2 above.

4.3.2. Distributed Molecules. The structure of the CM-200 distributed molecules implementation is essentially the same as that on the MasPar machine, with the extra level of complexity described in section 4.2.3 and with PARIS calls again being used to obtain maximum performance.

5. EXPERIMENTAL DETAILS AND RESULTS

5.1. Introduction. Similarity searches were carried out on a file of 2000 3-D molecules represented by interatomic distance matrices that had been generated from CONCORD structures.[35] Forty target molecules were chosen, varying in size from 6 to 33 non-hydrogen atoms. Four molecules of each size were chosen, two of which had a high similarity with at least some of the molecules in the data set, while the maximum similarities for the other two molecules at each size were much less; in fact, our experiments showed subsequently that there was only a negligible difference in CPU times between the high-similarity and low-similarity molecules.

The mean CPU times, when averaged over the four target molecules at each size, were obtained using the three parallel machines and also using a serial Evans and Sutherland ESV-30 UNIX workstation running at *ca.* 25 MIPS (MIPS = million instructions per second). Following the recommendations of Parkinson and Liddell,[36] we have measured the performance of the parallel similarity searches using the *speedup*. This is defined to be

$$T_{\text{serial}}/T_{\text{parallel}}$$

where T_{serial} and T_{parallel} are the CPU times of the most efficient sequential algorithm implemented on a serial processor and of the algorithm under investigation on the parallel processor, respectively. In the present context, the basis of comparison was the original sort–merge version of the atom-mapping algorithm[21] running on the ESV-30.

The serial CPU times and the various parallel speedups are detailed in Table 1. Note that the distributed molecules results, where one molecule is stored in each PE, have been scaled from the set of 2000 molecules used in the experiments to a data set of size 4096 molecules that would utilize all of the available PEs.

We have considered only the CPU times for two reasons. Firstly, all of the three machines were running other jobs concurrently with our programs, and the machine loading will affect the observed run times to some extent. Secondly, much of the run time was the time taken to load the data from backing storage into the memory of the front-end machine and then into the memory of the back-end parallel processor; however, the precise amount of time required for this stage depended on the particular I/O subsystem that was available (e.g., the DAP run times would have been significantly reduced if a parallel disk array had been attached).

5.2. Results. If we consider the DAP-610 searches first, an inspection of the results in Table 1 shows that the distributed distances searches are slower than the ESV serial searches when small target molecules are used, although the latter's CPU times increase greatly as the number of target atoms increase. Some improvement over the serial version may be seen with the larger target molecules, where speedups of up to 10 are achieved.

The CPU times for the distributed atoms algorithm averaged about 30% less than for the distributed distances method, with a consequent improvement in the calculated speedups. Early work on the DAP[30] indicated that distributed atoms worked substantially better than this; however, the results here are not directly comparable with our earlier experiments since the latter used an upper bound procedure[14] to rank the structures prior to the parallel similarity calculations; the inclusion of upper bounds is discussed further in section 6. The greater efficiency of the distributed atoms algorithm, when compared with the distributed distances algorithm, may be mainly attributed to the computational simplicity of the algorithm; although there is an extra level of complexity here, there is no need for the expensive shifting operations that are required for the distributed distances algorithm. There is also the point, which has been mentioned previously, that there is an inherent inaccuracy in the distributed distances algorithm, which results in about 10% of the true nearest neighbors for the target molecules not being retrieved.

Turning now to the MP-1104 searches, Table 1 shows that the distributed atoms algorithm here gives the best speedups of all of the experiments detailed in the table. However, the distributed molecules speedups for this machine are much less and are often comparable to the two sets of DAP searches.

The results of the CM-200 searches are rather disappointing, given the power and sophistication of this machine and the impressive results that have been obtained with it in a wide range of applications (see, e.g., ref 37). The maximum speedups for the distributed atoms and distributed molecules algorithms were 9.9 and 0.9, respectively (so that all of the distributed molecules searches were slower than the ESV-30 searches). Runs were also carried out using the full set of 16384 PEs that are available in the CM-200, rather than just 4096 as with the other two processors studied here. This did, of course, lead to an increase in performance but the speedups were still less than those of the smaller MasPar machine; for example, the average speedup for the size-33 CM-200 distributed atoms searches with all of the PEs was 22.9, as against 27.9 for the comparable MasPar searches.

There are several reasons why the CM-200 performance was less than might have been expected. Firstly, this particular machine runs at 8 MHz, rather than the full speed of 12 MHz, resulting in a one-third decrease in the maximum possible performance. Secondly, our programs used C*, which has not been optimized for the Weitek processors that are used in the CM-200; the Fortran compiler, conversely, has been so optimized and thus produces code that can be about twice as fast as the corresponding C* code. Finally, an individual CM-200 PE has an integer processing power of 0.5 MIPS (or 0.75 MIPS if the machine was run at 12 MHz), whereas an individual MasPar PE has an integer processing power of 1.6 MIPS, a substantial difference for an application such as this which is dominated by integer processing and which makes only limited use of floating-point operations. We are currently carrying out a detailed analysis to determine which of these factors are responsible for the generally-poor performance, in particular the performance of the distributed molecules algorithm.

6. DISCUSSION AND CONCLUSIONS

In this paper, we have studied the implementation of 3-D similarity searching using the atom-mapping method on massively-parallel array processors using three different algorithms—distributed distances, distributed atoms, and distributed molecules—on three of the most widely used types of SIMD machine—the DAP-610, the MasPar MP-1104, and the Connection Machine CM-200.

The DAP implementation of the distributed distances algorithm is quite slow, owing to the programming model that is used on this machine. The alternative models on the other two processors might enable this algorithm to be implemented more efficiently, but they could still do nothing to eliminate the incorrect results that can be given by this algorithm. Outer-loop algorithms, such as distributed molecules, are normally expected to provide the best level of performance on massively-parallel machines since they can make most use of the parallelism that is available. However, our results suggest that the speedups obtained with this algorithm are less than those obtained with distributed atoms (for reasons that are discussed below), and we thus suggest that the latter algorithm is the most appropriate way of implementing atom mapping in a massively-parallel environment, with the MasPar implementation giving speedups of about 30 times over a medium-power UNIX workstation.

So far, we have considered only the actual speedups that were obtained. However, it is also useful to have some measure of the potential speedup that could be achieved under ideal conditions. Probably the most widely used way of doing this is by means of *Amdahl's Law*,[38] which provides such a measure based on the proportion of the serial program that is inherently parallelizable. However, this is intended for use with parallel systems in which there is little difference in the processing speeds of the serial and parallel processors, as is generally the case with conventional multiprocessor systems and with MIMD (multiple instruction-stream, multiple data-stream) parallel machines. This is not the case with a massively-parallel machine where each PE is typically quite slow in comparison with a conventional serial processor. A more useful measure is hence the *scaled speedup*,

$$N \times \frac{S_{\text{parallel}}}{S_{\text{serial}}}$$

where N is the number of processors in the parallel machine, and S_{serial} and S_{parallel} are the processor speeds for the serial processor and for an individual PE in the parallel processor, respectively. This allows the calculation of the maximum-possible speedup for an outer-loop algorithm (i.e., using the serial algorithm on large numbers of data records in parallel, as in the distributed molecules method). Applying the scaled speedup to the ESV serial and, for example, the MasPar MP-1104 machine, we obtain a maximum speedup for distributed molecules of about 250, which is an order of magnitude greater than the actual results that were obtained (as detailed in Table 1). The main reason for this shortfall, and also for the fact that the distributed molecules algorithm is slower on both the MP-1104 and the CM-200 than distributed atoms, is the low *processor utilization*.[23] Scaled speedup assumes that all of the PEs are in use for all of the time. However, as we discussed in section 2, an essential feature of an SIMD machine is the use of masking to disable individual PEs: the processor utilization is simply the mean fraction of the PEs that are enabled. This fraction can be quite low in an algorithm like distributed molecules, since the time needed to process the set of distances associated with a target structure will be determined by the *maximum* of all of the times taken to process the distances associated with the individual database structures (since all of these structures are processed in parallel). The running time for distributed molecules is determined primarily by the number of atoms in a database structure, since each one of these is processed in turn; a single, large database structure that takes a long time to run to completion can thus substantially reduce the overall speed of searching (even though all of the other molecules have already been processed and their PEs must remain idle until the slow-running molecule has been processed completely).

The dependency of the search time on individual database records is an inherent characteristic of outer-loop algorithms. This is not a problem with fixed-format records, such as those that occur in conventional DBMS processing, but can result in significant inefficiencies when applied to unformatted databases, where individual records are of different sizes (or compositions). We first observed this phenomenon in searches of textual databases[33] and have also noted the effect when the DAP is used for 2-D substructure searching.[29] There are two ways in which the process utilization could be increased and in which we could thus improve the speedup. The first approach would be to rank the database in order of molecular size, so that molecules with comparable numbers of atoms would be processed at the same time, thus reducing the variation in the run times that would be expected for the set of structures currently occupying the PEs. Alternatively, it might be possible to hold the data for several molecules (or atoms if one wished to improve the processor utilization for the distributed atoms algorithm) on one PE, depending on the amount of local PE memory that was available, so that the amount of data being processed was better balanced across the PEs.

Previous work on atom mapping used *upper bounds* to increase the performance of the serial version of this similarity-searching technique.[14] It was found that a simple upper-bound calculation based on the molecular formulas of the target structure and each of the database structures was sufficient to eliminate a large fraction of a data set from the detailed atom-by-atom comparisons required in an atom-mapping search. This upper-bound calculation would be carried out far more efficiently on a massively-parallel machine than on a serial machine, using an outer-loop implementation in which sets of upper bounds are calculated in parallel, and then only the top-ranking structures would need to undergo the full atom-mapping algorithm using the distributed atoms implementation described above. We thus believe that a massively-parallel SIMD architecture is well suited to 3-D similarity searching using atom mapping; it will be of interest to see whether comparable results can be obtained using an MIMD architecture.

ACKNOWLEDGMENT

We thank the Science and Engineering Research Council for the award of a Research Studentship to D.J.W., Tripos Associates for the provision of the Evans and Sutherland ESV-30 workstation, Zeneca Agrochemicals for the provision of the file of structures, Catherine Pepperrell for the initial, serial implementation of atom mapping, and Geoffrey Downs for helpful comments on an initial draft of this paper. We are grateful to the following people for the provision of parallel computing facilities and much helpful advice: John Steel and his colleagues at the Centre for Parallel Computing, Queen Mary and Westfield College, University of London; Janet Wilkins at the University of Cambridge Computing Services; Richard Green, Nigel Jagger, and Victor Knight at MasPar Computer Ltd. (Reading, U.K.); David Baker at the Edinburgh Parallel Computing Centre, University of Edinburgh; and Harvey Richardson of Thinking Machines Corp. The Krebs Institute for Biomolecular Research is a designated center under the Molecular Recognition Initiative of the Science and Engineering Research Council.

REFERENCES AND NOTES

(1) Ash, J. E., Warr, W. A., Willett, P., Eds. *Chemical Structure Systems*; Ellis Horwood: Chichester, U.K., 1991.
(2) Willett, P. *Similarity and Clustering in Chemical Information Systems*; Research Studies Press: Letchworth, U.K., 1987.
(3) Johnson, M. A., Maggiora, G. M., Eds. *Concepts and Applications of Molecular Similarity*; Wiley: New York, 1990.
(4) Barnard, J. M.; Downs, G. M. Clustering of Chemical Structures on the Basis of Two-Dimensional Similarity Measures. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 644–649.
(5) Willett, P. *Three-Dimensional Chemical Structure Handling*; Research Studies Press: Taunton, U.K., 1991.
(6) Martin, Y. C. 3D Database Searching in Drug Design. *J. Med. Chem.* **1992**, *35*, 2145–2154.
(7) Dean, P. M. *Molecular Foundations of Drug-Receptor Interaction*; Cambridge University Press: Cambridge, U.K., 1987.
(8) Carbo, R.; Leyda, L.; Arnau, M. How Similar is a Molecule to Another? An Electron Density Measure of Similarity Between Two Molecular Structures. *International J. Quantum Chem.* **1980**, *17*, 1185–1189.
(9) Kearsley, S. K.; Smith, G. M. An Alternative Method for the Alignment of Molecular Structures: Maximising Electrostatic and Steric Overlap. *Tetrahedron Comput. Methodol.* **1990**, *3*, 615–633.
(10) Hermann, R. B.; Herronn, D. K. OVID and SUPER: Two Overlap Programs for Drug Design. *J. Comput.-Aid. Mol. Des.* **1990**, *5*, 511–524.
(11) Papadopoulos, M. C.; Dean, P. M. Molecular Structure Matching by Simulated Annealing. IV. Classification of Atom Correspondences in Sets of Dissimilar Molecules. *J. Comput.-Aid. Mol. Des.* **1991**, *5*, 119–133.
(12) Mezey, P. G. Shape-Similarity Measures for Molecular Bodies: a Three-Dimensional Topological Approach to Quantitative Shape–Activity Relations. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 650–656.
(13) Brint, A. T.; Willett, P. Upperbound Procedures for the Identification of Similar Three-Dimensional Chemical Structures. *J. Comput.-Aid. Mol. Des.* **1988**, *2*, 311–320.
(14) Pepperrell, C. A.; Taylor, R.; Willett, P. Implementation and Use of an Atom Mapping Procedure for Similarity Searching in Databases of 3-D Chemical Structures. *Tetrahedron Comput. Methodol.* **1990**, *3*, 575–593.
(15) van Geerestein, V.; Perry, N. C.; Grootenhuis, P. D. J.; Haasnoot, C. A. G. 3D Database Searching on the Basis of Ligand Shape Using the SPERM Prototype Method. *Tetrahedron Comput. Methodol.* **1990**, *3*, 595–613.
(16) Meyer, A. M.; Richards, W. G. Similarity of Molecular Shape. *J. Comput.-Aid. Mol. Des.* **1991**, *5*, 427–439.
(17) Bemis, G. W.; Kuntz, I. D. A Fast and Efficient Method for 2D and 3D Molecular Shape Description. *J. Comput.-Aid. Mol. Des.* **1992**, *6*, 607–628.
(18) Fisanick, W.; Cross, K. P.; Rusinko, A. Similarity Searching on CAS Registry Substances. 1. Global Molecular Property and Generic Atom Triangle Geometric Searching. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 664–674.
(19) Good, A. C.; Hodgkin, E. E.; Richards, W. G. The Utilization of Gaussian Functions for the Rapid Evaluation of Molecular Similarity. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 188–191.
(20) Nilakantan, R.; Bauman, N.; Venkataraghavan, R. New Method for Rapid Characterization of Molecular Shapes: Applications in Drug Design. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 79–85.

(21) Pepperrell, C. A.; Willett, P. Techniques for the Calculation of Three-Dimensional Structural Similarity Using Inter-Atomic Distances. *J. Comput.-Aid. Mol. Des.* **1991**, *5*, 455–474.

(22) Pepperrell, C. A.; Poirrette, A. R.; Willett, P.; Taylor, R. Development of an Atom Mapping Procedure for Similarity Searching in Databases of Three-Dimensional Chemical Structures. *Pestic. Sci.* **1991**, *33*, 97–111.

(23) Hwang, K.; Briggs, F. A. *Computer Architecture and Parallel Processing*; McGraw-Hill: New York, 1984.

(24) Hockney, R. W.; Jesshope, C. R. *Parallel Computers 2. Architecture, Programming and Algorithms*; Adam Hilger: Bristol, U.K., 1988.

(25) Trew, A.; Wilson, G. *Past, Present, Parallel. A Survey of Available Parallel Computing Systems*; Springer Verlag: London, 1991.

(26) Lewis, T. G.; El-Rewini, H.; Kim, I.-K. *Introduction to Parallel Computing*; Prentice Hall: Englewood Cliffs, NJ, 1992.

(27) Parkinson, D., Litt, J., Eds. *Massively Parallel Computing with the DAP*; Pitman: London, 1990.

(28) Rasmussen, E. M.; Willett, P.; Wilson, T. Chemical Structure Handling Using the Distributed Array Processor. In *Chemical Structures 2. The International Language of Chemistry*; Warr, W. A., Ed.; Springer Verlag: Berlin, 1993; pp 327–341.

(29) Willett, P.; Wilson, T.; Reddaway, S. F. Atom-by-Atom Searching Using Massive Parallelism. Implementation of the Ullmann Subgraph Isomorphism Algorithm on the Distributed Array Processor. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 225–233.

(30) Artymiuk, P. J.; Bath, P. A.; Grindley, H. M.; Pepperrell, C. A.; Poirrette, A. R.; Rice, D. W.; Thorner, D. A.; Wild, D. J.; Willett, P. Similarity Searching in Databases of Three-Dimensional Molecules and Macromolecules. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 617–630.

(31) Wild, D. J. Ph.D. thesis: University of Sheffield. Manuscript in preparation.

(32) Sedgewick, R. *Algorithms*, 2nd ed.; Addison-Wesley: New York, 1988.

(33) Willett, P.; Rasmussen, E. M. *Parallel Database Processing: Text Retrieval and Cluster Analysis Using the DAP*; Pitman: London, 1990.

(34) *The Connection Machine CM-200 Series Technical Summary*; Thinking Machines Corp.: Cambridge, MA, 1991.

(35) Rusinko, A.; Skell, J. M.; Balducci, R.; McGarity, C. M.; Pearlman, R. S. CONCORD: a Program for the Rapid Generation of High Quality Approximate 3-Dimensional Molecular Structures; University of Texas at Austin. This program is distributed by Tripos Associates, St. Louis, MO.

(36) Parkinson, D.; Liddell, H. M. The Measurement of Performance on a Highly Parallel System. *IEEE Trans. Comput.* **1983**, *C-32*, 32–37.

(37) Waltz, D. L. Applications of the Connection Machine. *Computer* **1987**, *20* (1), 85–96.

(38) Amdahl, G. M.; Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities. *AFIPS Conf. Proc.* **1967**, *30*, 483–485.