BIOINFORMATICS

Vol. 20 no. 12 2004, pages 1940–1947
doi:10.1093/bioinformatics/bth184

A case study of high-throughput biological data processing on parallel platforms

D. Pekurovsky[1], I. N. Shindyalov[1] and P. E. Bourne[1,2,*]

[1]San Diego Supercomputer Center and [2]Department of Pharmacology, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA

Received on August 31, 2003; revised on February 4, 2004; accepted on February 7, 2004
Advance Access publication March 25, 2004

ABSTRACT

Motivation: Analysis of large biological data sets using a variety of parallel processor computer architectures is a common task in bioinformatics. The efficiency of the analysis can be significantly improved by properly handling redundancy present in these data combined with taking advantage of the unique features of these compute architectures.

Results: We describe a generalized approach to this analysis, but present specific results using the program CEPAR, an efficient implementation of the Combinatorial Extension algorithm in a massively parallel (PAR) mode for finding pairwise protein structure similarities and aligning protein structures from the Protein Data Bank. CEPAR design and implementation are described and results provided for the efficiency of the algorithm when run on a large number of processors.

Availability: Source code is available by contacting one of the authors.

Contact: bourne@sdsc.edu

INTRODUCTION

Availability of high performance computing (HPC) architectures, including commodity clusters and traditional supercomputers with both distributed and shared memory designs, is becoming quite common. The massively parallel computation offered by these architectures has proven to be an important tool in a number of bioinformatics projects over the years. For a non-exhaustive list of examples, see Bhandarkar *et al*. (1996), Bourne and Hendrickson (1988), Brower and DeLisi (1992), Carvalho *et al*. (2000), Ceron *et al*. (1998), Date *et al*. (1993), Fekete *et al*. (2000), Foulser and Core (1990), Grundy *et al*. (1996), Jones (1992), Martino *et al*. (1994), Miller *et al*. (1991, 1992), Ropelewski *et al*. (1997, 2000), Shapiro *et al*. (2001) and Wallin *et al*. (1993).

Even with an abundance of HPC resources it is important to ensure their optimal use since many problems in bioinformatics deal with the analysis of large amounts of data and can often challenge even the most powerful HPC systems. As examples, consider genome annotation (Li *et al*., 2003), the comparison of a single open reading frame against a whole and relatively static annotated sequence database, and pairwise sequence and structure comparisons. The data being used for this class of comparative analysis problem (sequence–sequence, sequence–structure and structure–structure) are generally highly redundant. Thus, the data preprocessing step and the organization of the data being passed to these parallel computer architectures is of great importance if these data are to be processed in the most efficient manner. This paper addresses these issues with a specific example of pairwise structure–structure comparison. The analysis and conclusions contained in this paper apply to any distributed-memory platform.

*To whom correspondence should be addressed.

Problem statement

The determination and subsequent three-dimensional (3D) comparison of protein structures is important for reasons of protein classification, a better understanding of function, and the elucidation of distant homologous relationships not possible from sequence alone, since sequence is more variable than structure (Rost, 1999). In recognition of the importance of structure, a number of structural genomics projects are under way worldwide (Zhang and Kim, 2003). These are large-scale efforts to determine, or subsequently infer by modeling, protein structures for most discovered genes. Thus the current (February 2, 2004) complement of protein structures (24 080) is likely to expand significantly in the next few years as a result of structural genomics as well as a more concerted effort in conventional functionally-driven structure determination. Comparative analysis of this expanding corpus represents a challenge to those engaged in high performance computing application development.

Many structure comparison algorithms have been developed in the last 20 years [for a review, see Holm and Sander (1994), Gibrat *et al*. (1996) and Godzik (1996)]. Only a few of these algorithms have become resources available through the Internet and are updated on a regular basis, in part a result of the large and on-going computational analysis required. These algorithms can be classified by the elementary units used in comparison to build the alignment: (1) single residues (Orengo *et al.*, 1992); (2) fragments of multiple residues (Holm and Sander, 1993; Feng and Sippl, 1996; Shindyalov and Bourne, 1998) and (3) secondary structure elements (Madej *et al.*, 1995; Alexandrov and Go, 1994). They can also be classified by the optimization procedure used: (1) dynamic programming (Orengo *et al.*, 1992); (2) Monte-Carlo (Holm and Sander, 1993); (3) combinatorial search (Shindyalov and Bourne, 1998) and (4) graph theory (Alexandrov and Go, 1994).

This work optimizes the use of the Combinatorial Extension (CE) algorithm for the pairwise alignment of polypeptide chains proposed earlier (Shindyalov and Bourne, 1998), and the Protein Object Model (POM; Shindyalov and Bourne, 1997) for managing comparative structural information. The problem addressed by this paper is building a structurally representative set of protein chains and revealing structure similarities in the PDB that will scale with a fast growing source of data. Hence, the algorithm for the calculation of this representative set should give optimal performance and also scale with the number of processors involved. The availability of the representative set allows users to save time and computational resources when comparing the structure of their protein chains against those in the PDB without a significant loss of information. This paper does not address the details of the CE algorithm itself, which are well covered elsewhere (see Shindyalov and Bourne, 2001) and references therein), nor the issue of comparison of domains versus polypeptide chains, which is the subject of a separate ongoing study.

To illustrate the scale of the problem, consider a PDB of 35 000 chains with each pairwise comparison taking on average ∼3 s. Without considering redundancy or chain size a complete computation would take ((35 000∗35 000)/2)∗3 s or approximately 21 000 processor-days. Given this overall time, optimization with regard to the data is clearly necessary.

SYSTEM AND METHODS

CE overview

The CE algorithm is a method of automatically aligning pairs of structures, previously described in Shindyalov and Bourne (1998). The CE algorithm compiles an alignment of a given pair of protein chains by considering the chains sectioned into all possible octapeptide fragments, as defined by the backbone α-carbons. Those octapeptide pairs that have a high distance-based similarity score are deemed aligned fragment pairs (AFPs) and retained for the next step in the analysis. CE then tries to join each AFP to a maximal number of other AFPs in order to create the longest possible alignment path through the two proteins in consideration (with an allowance for gaps of up to 30 residues in either protein chain). In effect this 'stitches together' a set of AFPs covering contiguous regions in each protein that represents a possible alignment path. Once the possible paths through the two proteins are determined, CE uses additional heuristics in an attempt to improve the final alignment. The 20 best scoring paths are compiled and the proteins are directly compared based upon the superimposition of the aligned residues. The path that yields the lowest RMSD is then retained as the 'optimal path'. This path is then subjected to dynamic programming on structural alignment directly between the two structures, which tests all possible residue equivalences and the resulting RMSD from their superposition.

Compute platform

The computational part of this work was performed using 'Blue Horizon', an IBM SP parallel computer installed at the San Diego Supercomputer Center (SDSC). It has a total of 1152 Power3+ processors, each running at 375 MHz. The software is equally suited for running on any parallel machine or PC cluster equipped with an implementation of the Message Passing Interface (MPI). In addition to IBM SP, we have tested the software on a Sun Enterprise 10 000 server and a Linux PC cluster.

ALGORITHM AND IMPLEMENTATION

CEPAR, the algorithm described here, is a version of the CE algorithm designed specifically for use on parallel (PAR) processors.

Problem formulation

Given:

(1) An entity list of N entities (each entity being a protein polypeptide chain characterized by an amino acid sequence and a set of 3D coordinates).

(2) An algorithm for pairwise comparison of entities (in this case CE).

(3) Representation criteria whereby one entity represents a family of one or more entities, based on the results of a pairwise alignment. Here this implies
   (a) $|L_1 - L_2| < L_{\text{thr}}(L_1 + L_2)/2$, where L_1 and L_2 are sequence lengths of the two entities, and L_{thr} is the length difference threshold parameter.
   (b) $L_{\text{ali}} \geq A_{\text{thr}}(L_1 + L_2)/2$, where L_{ali} is the number of aligned positions and A_{thr} is the alignment length threshold parameter.
   (c) $L_{\text{gap}} < G_{\text{thr}} L_{\text{ali}}$, where L_{gap} is the number of residues in gaps and G_{thr} is the gap threshold parameter.
   (d) The final RMSD of the alignment RMSD $\leq R_{\text{thr}}$, where R_{thr} is the RMSD threshold parameter.

(4) Similarity criterion between representatives. Here, this implies a CE Z-score $Z \geq Z_{\text{thr}}$ where Z_{thr} is the Z threshold parameter [for a description of Z-score see (Shindyalov and Bourne, 1998)]. Alignments not satisfying this criterion are not recorded.

Desired output: a list of representatives as well as those entities represented by them and detailed information on alignments satisfying either representation or similarity criterion[1].

Establishing a representative is not exactly the same as applying a rigorous clustering in the sense of vector quantization. In this work, the representatives are randomly chosen instead of calculating the centroid of a cluster. Vector quantization is not carried out so as to minimize computer time. The representation criteria applied is believed to adequately describe the structural space of the PDB.

Parallel algorithm

Rather than focus on the parallelization of the CE algorithm for each pairwise alignment, CEPAR uses a coarse-grain parallel implementation involving a master/worker strategy suitable for a massively parallel computer architecture. Each worker, upon receiving a work assignment from the master, compares the two entities contained in the assignment using the CE algorithm, returns the results of the comparison to the master and is ready to receive another assignment.

More generally, more than one command processor could be employed. However, for the problem considered here this is not advantageous as will be explained in the following subsections. Therefore, for the work presented in this paper one master processor was used.

Note that the algorithm is not truly, embarrassingly, parallel since there are dependencies in the order of distributing assignments, as will become more apparent below. The program is written in C++ and uses MPI for communication between the master and workers. The workers need to communicate only with the master processor not each other. Subsequent discussion centers on the work of the master processor.

Order of operations

We have considered and compared two strategies for the order of operations. Both start by making the first entity in the input data the first representative, and forming a list of all remaining entities (the entity list). In the entity-first approach, the strategy is to go through the entity list aligning each entity with the first representative, one entity per processor. The entities matching the representative (according to the criterion mentioned above) are deleted from the entity list and assigned to the representative's group. After all entities in the entity list have been exhausted the first entity in the remaining list is made the second representative, is deleted from the list and the process repeated. Thus at any stage of the calculation we are attempting to exhaust alignments of all remaining entities in the list with a given representative before proceeding to the next representative[2].

[1]Based on the saved alignment information and known representation it is easy to restore alignments of each entity with any other entity in the PDB. This is done as a post-processing step and is not included in this discussion.

[2]In parallel mode, it may be sometimes necessary for the program to be aligning each entity with more than one representative at a time, since the number of processors available may be greater than the number of entities remaining on the list. In this case the algorithm starts resembling the family-first approach.

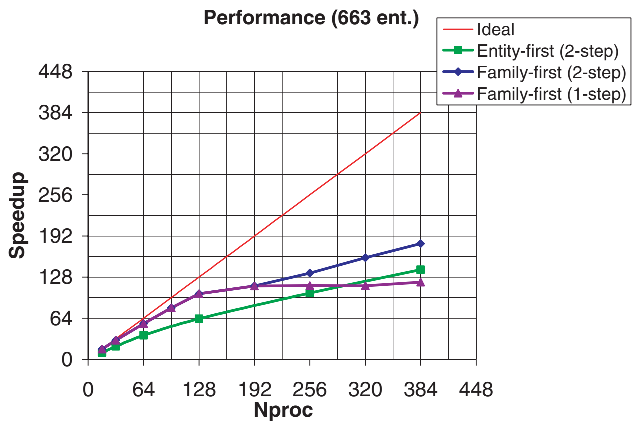

Fig. 1. Comparative performance of two parallel versions of CEPAR: family-first and entity-first (green squares) strategies. The family-first implementation was run using both one-step (magenta triangles) and two-step (blue diamonds) jobs. Sample data consisting of 663 entities chosen at random from the PDB were used to obtain these measurements. 321 representatives were identified using the following threshold parameters: $L_{\text{thr}} = 10\%$, $A_{\text{thr}} = 67\%$, $G_{\text{thr}} = 20\%$, $R_{\text{thr}} = 2$ A, $Z_{\text{thr}} = 3.5$. Ideal scaling is shown in red. The vertical axis plots the speedup relative to the 16-processor timing of 5567 s for the family-first approach (i.e. 16 units = 1/1.55 h). 20% average idle time was used as a stopping condition for the first step of the two-step run (described in the text). The timing used to plot the two-step runs is the combined CPU hours of the two steps divided by the number of processors in the first step. Note: the scaling curve in this plot applies only to the performance of CEPAR with a small sample data set. For realistic data set sizes the scalability is more extensive.

In an alternative family-first approach, the first entity from the entity list (referred to as the current entity) is aligned in turn with all representatives identified so far, one representative per processor, before going on to the next entity on the list. If a relationship is established with one of the representatives, the current entity is assigned to its group, is retired from all ongoing computations and the next entity becomes current. If all possible alignments of existing representatives with the current entity are exhausted without establishing a relationship, the current entity becomes a new representative and the process repeats with the next entity on the list. In parallel mode, it may be necessary to consider more than one current entity at a time, since the number of processors may be greater than the number of existing representatives. Here the term 'family' does not necessarily imply a biological relationship, but rather a notion of a representative of a group of proteins based on criteria associated with likeness of 3D structure.

For the given criteria used to define a match, the family-first strategy was superior both in performance (Fig. 1) and

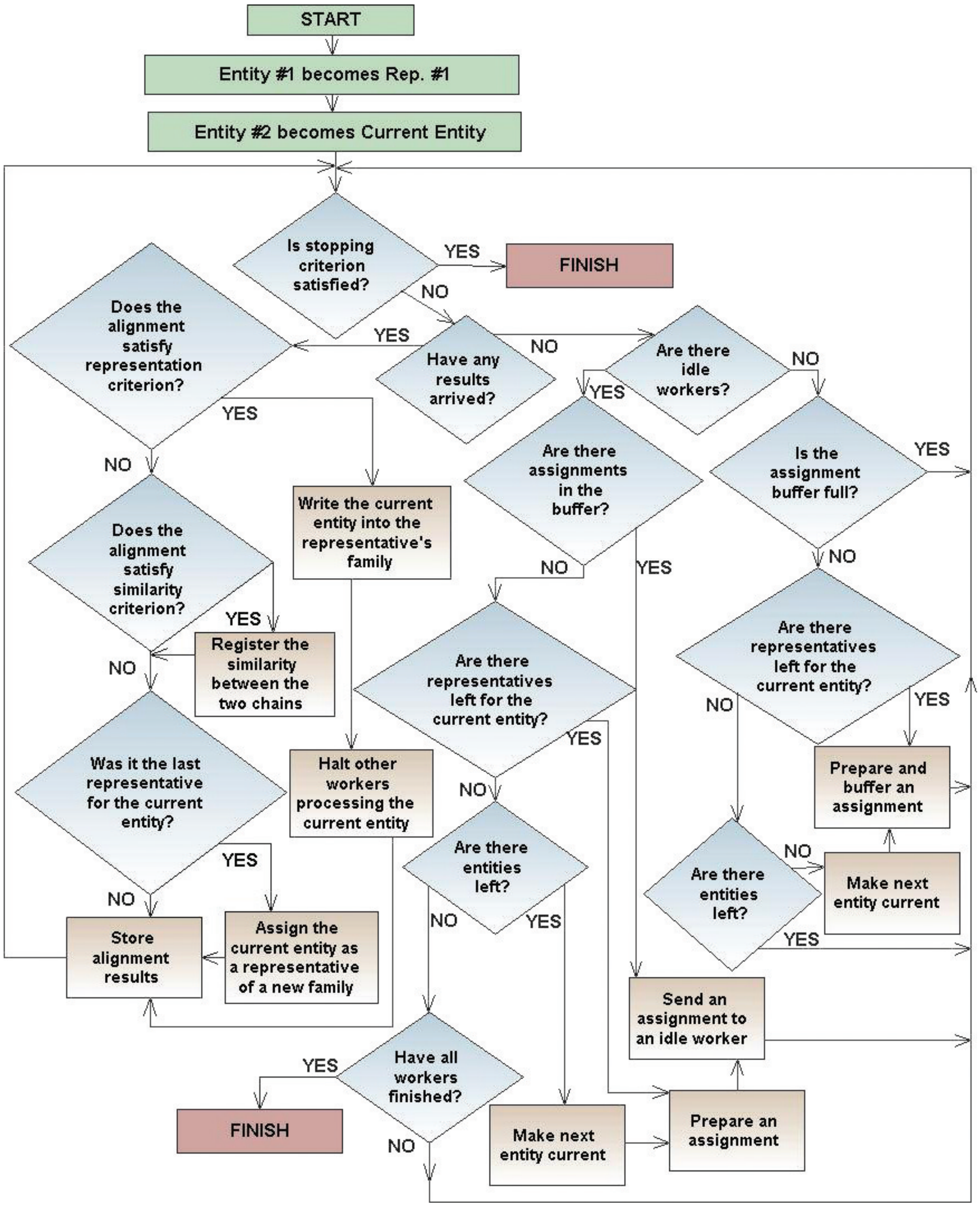
START
Entity #1 becomes Rep. #1
Entity #2 becomes Current Entity
Is stopping criterion satisfied?
YES
FINISH
NO
Does the alignment satisfy representation criterion?
YES
Have any results arrived?
NO
Are there idle workers?
NO
YES
Are there assignments in the buffer?
Is the assignment buffer full?
YES
NO
Write the current entity into the representative's family
Does the alignment satisfy similarity criterion?
YES
Register the similarity between the two chains
NO
Was it the last representative for the current entity?
YES
NO
Store alignment results
Assign the current entity as a representative of a new family
Halt other workers processing the current entity
Are there representatives left for the current entity?
NO
YES
Are there entities left?
NO
YES
Have all workers finished?
YES
FINISH
NO
Make next entity current
Prepare an assignment
Send an assignment to an idle worker
Are there representatives left for the current entity?
NO
YES
Prepare and buffer an assignment
Are there entities left?
NO
YES
Make next entity current

Fig. 2. Algorithm logic for the master processor using the family-first strategy.

in memory utilization to the entity-first strategy. This is discussed subsequently. From this point on we focus on the family-first approach. Figure 2 shows the logic used for the master processor when executing the family-first strategy.

Scalability bottlenecks

From Figure 1, it is clear that running CEPAR in one step as described produces limited scalability. By tracing the number of idle workers and time taken for communication operations the limited scalability at high processor counts is a result of load imbalance at the end of the run. At this point most of the worker processors run out of tasks while only a few finish their last assignment. Resource reservation systems on most public supercomputers reserve a block of processors, making it impossible to release them one by one. Therefore, in production mode the number of processors assigned should not be more than is reasonable, namely $N_p < N_{th}$, where the threshold number, N_{th}, is several times less than the number of entities to be processed. An alternative approach is to run the program in two steps, utilizing an early stopping condition, which, for example, causes the first of the two runs to abort when accumulated average idle time of workers exceeds a predefined amount, such as 20% of the total run time. The remaining part of the calculation is then completed on a smaller number of processors. The result of using a two-step approach is discussed subsequently.

Now consider purely the parallel efficiency of CEPAR for the duration of the first step of the run, ignoring the end-of-run effects. Parallel efficiency starts to decrease at a sufficiently high processor count as the master processor becomes resource bound while trying to supply workers with their assignments. While the time required to create new assignments is small, it is nevertheless finite. Therefore, with an increasing number of workers there is a higher chance of congestion. This condition applies to both entity-first and family-first strategies.

One way to avoid such congestion would be to consider variants of the algorithm other than the basic master/workers approach. For example, one or more processors other than the master might receive and analyze results from the workers. However, this approach will not work well for the problem we are considering. Keep in mind that the results of each task (i.e. whether an entity matches a representative) has a substantial influence on the choice of subsequent assignments if one desires to avoid redundant computations (see below). This means that in order to preserve the highest efficiency of the algorithm it would be necessary to introduce communication between the processor(s) handling the distribution of assignments and the processor(s) collecting the results. That is, congestion of the master processor would not be reduced by introducing other command processors. On the other hand, if communication between the command processors is designed to be less than regular, there will be less idle time for workers but they will often perform redundant operations and that will negate the overall increase in parallel performance.

Therefore, based on the combined considerations of redundancy in assignments and master processor congestion, we implemented only the basic master/workers algorithm. We did introduce, however, several measures that significantly alleviate the master processor congestion. First, we implemented advanced buffering (or queuing) of assignments performed when the master processor is idle. Advanced buffering requires identifying appropriate entity/representative pairs and packing them in the form of packets of MPI_Packed type that can be dispatched to the workers without any delay. The packets are queued and sent out in order. Second, decreasing the amount of disk I/O and standard single-CPU optimization techniques allowed us to reduce the time it takes the master processor to prepare assignments and analyze results. These measures led to a decrease in the master congestion so the resulting scalability is more than adequate for this work.

The following is a general description of the family-first algorithm as run on the master processor in a wait loop fashion (Fig. 2). The master processor reads data from the disk, then prepares and sends assignments to workers. Assignment packets consist of amino acid sequences and C_α atomic coordinates for the current entity and one of the representatives. The assignments are distributed on a first come first served basis, since the time of individual task completion varies. The master checks for incoming messages from workers as often as possible, since any incoming message can influence the course of further assignments. As soon as a message arrives (containing completed results from one of the workers), the master proceeds to assess the results and send the next assignment to one of the idle workers. In cases when the entity is determined to belong to an existing representative, the master also halts other workers processing the same entity. When all workers have received their tasks and no incoming messages are in sight, the master processor prepares a few more assignments and stores them in the queue in order to have them ready to be dispatched without delay.

Clearly, if there is a match between the representative and the entity, we would wish to find it as soon as possible to avoid initiating redundant task assignments with the same entity. Therefore, it is important to sort the representatives in decreasing order of chance of being similar to the given entity. We estimate this chance by, first, giving priority to those representatives having a number of residues within 10% of the current entity, and second, by using similarity in amino acid content based on frequency profiles. This approach is approximate but provides performance gains over a random or sequential choice of representatives.

MPI communication

At first glance, the efficiency of MPI communication appears to play an insignificant role in overall performance since communication time is a small fraction of the overall CEPAR computation time. Nevertheless, care must be taken in using the most appropriate MPI send function for the hardware and software in hand. For example, in the case of IBM's implementation of MPI, the blocking send function `MPI_Send()` is inappropriate because this implementation does not buffer the messages for large message sizes (Barrios *et al.*, 1999, http://www.redbooks.ibm.com). Other vendors supporting MPI may implement a buffered version of `MPI_Send()`. MPI implementations that avoid buffering messages can cause deadlock in some cases [for a detailed discussion of this,

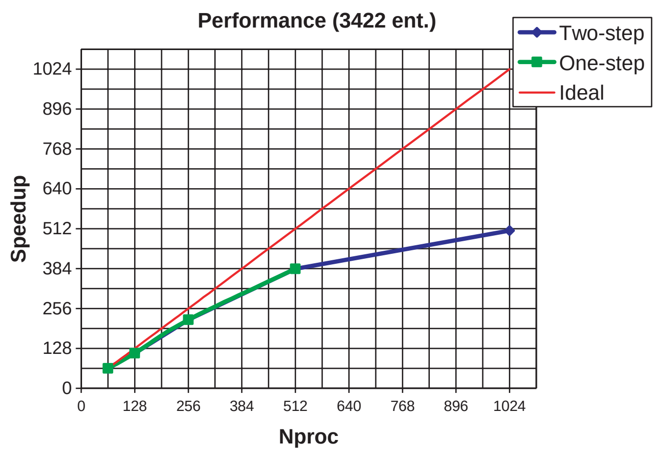

Fig. 3. Parallel performance of CEPAR run on a data set of size 3422 entities where 1159 families were identified using the following threshold parameters: $L_{\text{thr}} = 10\%$, $A_{\text{thr}} = 67\%$, $G_{\text{thr}} = 20\%$, $R_{\text{thr}} = 2$ A, $Z_{\text{thr}} = 3.5$. Green (squares) correspond to one-step runs, while blue (diamonds) correspond to two-step execution. The red line shows ideal scaling. Performance is plotted as speedup relative to the 64 processors run lasting 18948 s (i.e. 64 units = 1/5.26 h). 20% average idle time was used as a stopping condition for the first step of the two-step experiment. The timing used to plot the latter run is combined CPU hours of the two steps divided by the number of processors in the first step. Note: the scaling curve in this plot applies only to performance of CEPAR with a small, sample data set. For realistic data set sizes the scalability is even more extensive.

see sections 2.8.7, 2.13 and 9.2.1 in the MPI reference (Snir *et al.*, 1996) as well as section 3.1.2 in the IBM Redbook (Barrios *et al.*, 1999)]. In the case of CEPAR, no deadlock occurs, however, the master processor can be blocked while waiting for some worker processors to finish. The `MPI_Bsend()` function for buffered sends solves this problem.

RESULTS

The scalability of CEPAR (family-first implementation) was established from two sample databases of sizes 663 and 3422 entities, respectively (Figs 1 and 3). The one-step execution performance shown in Figure 1 illustrates that the code scales well at the lower end of processor count, but the efficiency curve flattens at processor counts several times less than the number of entities. Running the program in two steps indicates that most of the performance loss in one-step experiments is a result of the end-of-run load imbalance. As the size of the entity list increases this becomes less of a problem; thus for the full PDB, the end-of-run load imbalance is not significant. A suitable parameter for estimating the effect of the end-of-run load imbalance is the ratio of the input entity list (Nent) to the number of processors used in the run (N_p). According to Figures 1 and 3, the end-of-run load imbalance effect starts to become significant at Nent$/N_p = 7$–10, while in the production run across all the PDB the ratio defined above will be at least of the order of 30 or more. Thus, the end-of-run imbalance is a usability rather than performance issue. When it is desirable to avoid wasting even a small fraction of resources in a large-scale run, the program can be executed in two steps. To this end we have implemented an optional early stopping condition, as discussed earlier. The first step is aborted when the load imbalance starts to set in. The calculation is finished in a second step using a much smaller number of processors. The practical use of CEPAR was made easy by developing a utility for automatically processing a specified input set in either one or two steps.

Other than in the end of the run, performance of CEPAR is quite close to the peak of scalability within the studied range of processor count (up to 512 processors and beyond). The difference between the actual and ideal scaling is caused mostly by load imbalance due to congestion of the master processor as well as the redundancy of calculations of the same entity by several processors.

DISCUSSION AND SUMMARY

Bioinformatics is faced with highly redundant datasets requiring, in some cases, very large computation to be performed to gain insights into the meaning of these data. Fortunately, large compute systems, most frequently composed of collections of commodity processors, are now within the reach of departments and small organizations and capable of being used to analyze these data. Efficient use of these resources depends on meticulous design of the algorithms and software with performance and scalability given a high priority. Here, we have analyzed a particular application—pairwise 3D protein structure comparison—but the principles of what needs to be done to optimize an application are general and therefore worth discussing.

Our first finding relates to the organization of the data being fed to a large array of loosely coupled processors. Consideration of the way the data is processed is paramount to gaining good performance. Any handling of known redundancy and ordering of data according to obvious time dependencies must clearly be looked at first. Optimization of the algorithm for distribution of assignments was crucial in decreasing the master processor congestion and achieving reasonably good scalability.

Second, in the case of CEPAR, we established empirically that the family-first approach outperformed the entity-first approach. Intuitively, the reason for this advantage comes from the relative number of representatives (families) versus the total number of entities. This ratio changes as the computation progresses. In this application, the ratio overall was to favor the family-first approach since, overall, the likelihood that a new entity would match one of a relatively fast growing and small number of representatives out-performed the entity-first approach, which iteratively processes a large entity list. Note that memory restrictions on many compute platforms may dictate the use of the family-first approach as significantly more memory efficient. A reasonable approach (not tried) might be to switch from an entity-first to a family-first approach after a significant number of representatives have been found or the memory is getting close to being exhausted.

Our third finding relates to the end of run load imbalance and allocation of processors typical of HPC architectures. A large number of processors could be left in an idle state while the remaining work is performed on a relatively small number of assignments. The severity of this condition is dependant on the size of the input data set. We addressed this issue with an early stopping condition, performing the small number of remaining tasks on a small number of processors.

Finally, the use of MPI sends in distributing tasks in a master/worker environment requires careful consideration as MPI implementations vary on different hardware and software architectures.

In summary, when solving the family classification problem, CEPAR scales to 512 processors and beyond on the IBM SP 'Blue Horizon' supercomputer even on a sample data set one-tenth of the size of the PDB. CEPAR is an order of magnitude more efficient that the original CE code when run on a large number of processors. This is important as improvements are made to the original algorithm and a complete new pairwise comparison of a fast-growing PDB is needed. It also has merit for use during weekly updates of the PDB when new entities on the order of 100–200 need to be compared to the complete previous version of the PDB.

As outlined in the Introduction section, many algorithms have been developed for pairwise structure comparison, but few lead to weekly updated databases of complete comparisons. In part this results from the magnitude of the computations that must be undertaken. Optimized algorithms like that described here for the current generation of hardware are therefore important. The principles described here can be applied to different types of computations beyond a pairwise comparison of protein structures. The database of comparisons resulting from this work can be found at http://cl.sdsc.edu/ce.html

ACKNOWLEDGEMENTS

The work of D.P. was funded through the Molecular Sciences Trust of the National Partnership for Advanced Computational Infrastructure (NPACI), National Science Foundation (NSF). The work of I.N.S was funded by NSF DBI 9808706. The work of P.E.B. was funded by NIGMS 1P01GM63208-01A1. Parallel runs were made on computers at the SDSC.

REFERENCES

Alexandrov,N.N. and Go,N. (1994) Biological meaning, statistical significance, and classification of local spatial similarities in nonhomologous proteins. *Protein Sci*, **3**, 866–875.

Barrios,M. Andersson,S., Bhanot,G., Hague,J., Johnston,F., Kandadai,S., Klepalki,D., Levesque,J. Nieplocha,J., O'Connell,F. *et al*. (1999) *Scientific Applications in RS/6000 SP Environments*, IBM Redbooks series.

Bhandarkar,S.M., Chirravuri,S. and Arnold,J. (1996) Parallel computing of physical maps – a comparative study in SIMD and MMD parallelism. *J. Comput. Biol.*, **3**, 503–528.

Bourne,P.E. and Hendrickson,W.A. (1988) Selecting a processor for computations in molecular biophysics. *Comput. Biol. Med.*, **18**, 341–349.

Brower,R.C. and DeLisi,C. (1992) Impact of massively parallel computation on protein structure determination. *Crit. Rev. Biomed. Eng.*, **20**, 373–389.

Carvalho,A.P., Gomes,J.A. and Cordeiro,M.N. (2000) Parallel implementation of a Monte Carlo molecular stimulation program. *J. Chem. Inf. Comput. Sci.*, **40**, 588–592.

Ceron,C., Dopazo,J., Zapata,E.L., Carayo,J.M. and Trelles,O. (1998) Parallel Implementation for DNAml program on Message-Passing Architectures. *Parallel Comput.*, **24**, 701–716.

Date,S., Kulkarni,R., Kulkarni,B., Kulkarni-Kale,V. and Kolaskar,A.S. (1993) Multiple alignment of sequences on parallel computers. *CABIOS*, **7**, 237–247.

Fekete,M., Hofacker,I.L. and Stadler,P.F. (2000) Prediction of RNA base pairing probabilities on massively parallel computers. *J. Comput. Biol.*, **7**, 171–182.

Feng,Z. and Sippl,M. (1996) Optimum superimposition of protein structures: ambiguities and implications. *Fold. Des.*, **1**, 123–132.

Foulser,D.E. and Core,N.G. (1990) Parallel computation of multiple biological sequence comparisons. *Comput. Biomed. Res.*, **23**, 310–331.

Gibrat,J-F., Madej,T. and Bryant,S.H. (1996) Surprising similarities in structure comparison. *Curr. Opin. Struct. Biol.*, **6**, 377–385.

Godzik,A. (1996) The structural alignment between two proteins: is there a unique answer? *Protein Sci.*, **5**, 1325–1338.

Grundy,W.N., Bailey,T.L. and Elkan,C.P. (1996) ParaMEME: A Parallel Implementation and a Web Interface for a DNA and Protein Motif Discovery Tool. *CABIOS*, **12**, 303–310.

Holm,L. and Sander,C. (1993) Protein structure comparison by alignment of distance matrices. *J. Mol. Biol.*, **233**, 123–138.

Holm,L. and Sander,C. (1994) Searching protein structures has come of age. *Proteins*, **19**, 165–173.

Jones,R. (1992) Sequence pattern matching on a massively parallel computer. *CABIOS*, **8**, 377–383.

Li,W.W., Quinn,G.B., Alexandrov,N.N., Bourne,P.E. and Shindyalov,I.N. (2003) Proteins of *Arabidopsis thaliana* (PAT) database: a resource for comparative proteomics *Genome Biol.*, **4**, R51.

Madej,T., Gibrat,J-F. and Bryant,S.H. (1995) Threading a database of protein cores. *Proteins*, **23**, 356–369.

Martino,R.L. Johnson,C.A., Suh,E.B., Trus,B.L. and Yap,T.K. (1994) Parallel computing in biomedical research. *Science*, **256**, 902–908.

Miller,P.L., Nadkarni,P.M. and Carriero,N.M. (1991) Parallel computation and FASTA: confronting the problem of parallel database search for a fast sequence comparison algorithm. *CABIOS*, **7**, 71–78.

Miller,P.L., Nadkarni,P.M. and Bercovitz,P.A. (1992) Harnessing networked workstations as a powerful parallel computer: a general paradigm illustrated using three programs for genetic linkage analysis. *CABIOS*, **8**, 141–147.

Orengo,C.A., Brown,N.P. and Taylor,W.T. (1992) Fast structure alignment for protein databank searching. *Proteins*, **14**, 139–167.

Ropelewski,A.J., Nicholas,H.B., Jr, Deerfield,D.W., II (1997) Implementation of Genetic Sequence Alignment Programs on Supercomputers. *J. Supercomput.*, **11**, 237–253.

Ropelewski,A.J., Nicholas,H.B., Jr, Deerfield,D.W., II (2000) Selective and sensitive comparison of genetic sequence data on high performance computers. In A.Koniges (ed.), *Parallel Computing for Industrial Applications*. Morgan Kaufmann, pp. 453–479.

Rost,B. (1999) Twilight zone of protein sequence alignments. *Protein Eng.*, **12**, 84–95.

Shapiro,B.A., Wu,J.C., Bengali,D. and Potts,M.J. (2001) The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation. *Bioinformatics*, **13**, 137–148.

Shindyalov,I.N. and Bourne,P.E. (1997) Protein data representation and query using optimized data decomposition. *CABIOS*, **13**, 487–496.

Shindyalov,I.N. and Bourne,P.E. (1998) Protein structure alignment by incremental combinatorial extension of the optimum path. *Protein Eng.*, **11**, 739–747.

Shindyalov,I.N. and Bourne,P.E. (2001) CE: A resource to compute and review 3-D protein structure alignments. *Nucleic Acid Res.*, **29**, 228–229.

Snir,M., Otto,S., Huss-Lederman,S., Walker,D. and Dongarra,J. (1996) *MPI: The Complete Reference*. MIT Press, Cambridge, MA.

Wallin,E., Wettergren,C., Hedman,F. and von Heijne,G. (1993) Fast Needleman-Wunsch scanning of sequence databanks on a massively parallel computer. *CABIOS*, **9**, 117–118.

Zhang,C. and Kim,S-H (2003) Overview of structural genomics: from structure to function. *Curr. Opin. Chem. Biol.*, **7**, 28.