From: CABIOS 12:303-310

ParaMEME: A Parallel Implementation and a Web Interface for a DNA and Protein Motif Discovery Tool

William N. Grundy* email: bgrundy@cs.ucsd.edu
Department of Computer Science and Engineering (619) 453-4364
University of California at San Diego FAX (619) 534-7029
La Jolla, California 92093-0114

Timothy L. Bailey email: tbailey@sdsc.edu
San Diego Supercomputer Center (619) 534-8350
P.O. Box 85608 FAX (619) 534-5127
UPS, Fed Ex: 10100 Hopkins Drive
San Diego, California 92186-9784

Charles P. Elkan email: elkan@cs.ucsd.edu
Department of Computer Science and Engineering (619) 534-8897
University of California at San Diego FAX (619) 534-7029
La Jolla, California 92093-0114

March 4, 1997

*To whom reprint requests should be sent.

Abstract

Many advanced software tools fail to reach a wide audience because they require specialized hardware, installation expertise, or an abundance of CPU cycles. The worldwide web offers a new opportunity for distributing such systems. One such program, MEME, discovers repeated patterns, called motifs, in sets of DNA or protein sequences. This tool is now available to biologists over the worldwide web, using an asynchronous, single-program multiple-data version of the program called ParaMEME that runs on an Intel Paragon XP/S parallel computer at the San Diego Supercomputer Center. ParaMEME scales gracefully to 64 nodes on the Paragon with efficiencies above 72% for large data sets. The worldwide web interface to ParaMEME accepts a set of sequences interactively from a user, submits the sequences to the Paragon for analysis, and e-mails the results back to the user. ParaMEME is available for free public use at `http://www.sdsc.edu/CompSci/Biomed/MEME`.

Keywords: intelligent system, supercomputer, worldwide web, sequence analysis, parallel computing.

Running head: ParaMEME: Discovering Motifs with a Supercomputer

1 Introduction

State-of-the-art intelligent software tools have rarely reached the commercial mainstream because the technology life-cycle for such software is short and because the target audience is often small. When distributed at all, research software is usually placed on an ftp site for potential users to download and install. Such distributions often impose particular hardware and operating system constraints. They also require considerable operating systems expertise on the part of the user.

The worldwide web offers an alternative mode of software distribution. Rather than disseminating the software itself, the web can provide a gateway to a single system running the software at a central location, accessible by more than 7.5 million people in the United States alone [WSJ, 1996]. For the user, such a paradigm removes the difficulties involved in downloading and installing the system. It also provides access for inexperienced or occasional users and for users who lack the appropriate hardware or software to install the system at their own sites. For example, a server at at the Weizmann Institute of Science [BIOCCELERATOR, 1996] provides web access to the BIOCCELERATOR series of database search engines which run on special-purpose hardware. Also, an easily accessible web interface allows users who are considering installing the software locally to test the software before installing it. For the authors of the software, a web interface to their system allows for easy upgrades and bug fixes, as well as the opportunity to monitor usage of the system.

MEME is a software tool for discovering motifs in groups of DNA or protein sequences [Bailey and Elkan, 1995b; Bailey and Elkan, 1995a]. The acronym stands for "Multiple EM for Motif Elicitation," because the program uses the expectation-maximization (EM) algorithm to locate one or more repeated patterns, called "motifs," in a sequence or series of sequences. MEME takes as input one or more sequences believed to contain one or more motifs, and outputs a series of motif models. Each of these models is a position-dependent letter frequency matrix (similar to a gapless profile [Gribskov *et al.*, 1990]) and describes a single motif. Our definition of motifs is related to that of “blocks” [Henikoff *et al.*, 1995] — multiply aligned sequence segments that correspond to the most highly conserved regions of protein families — but is more general because we allow motifs to represent patterns that repeat in a single sequence, and because we consider DNA as well as protein motifs. Motifs are ideally suited for, among other things, database searches [Tatusov *et al.*, 1994], identifying structurally and functionally important portions of proteins [Bailey and Elkan, 1996], characterizing DNA protein-binding sites [Schneider *et al.*, 1986] and designing degenerate PCR primers [D’Esposito *et al.*, 1994]. MEME discovers motifs automatically given only a group sequences as input.

In general, discovering motifs is difficult because the patterns need not be exact; there may be only approximate similarity between instances of one motif. As the number of sequences increases, exhaustive search quickly becomes intractable. MEME differs from similar tools such as the Gibbs sampler [Lawrence *et al.*, 1993] in that MEME does not require the user to provide prior knowledge about the distribution of instances of motifs in the given data set. MEME can find motifs which are distributed unevenly through the data (e.g., three motifs in one sequence and zero in another), and can also determine their widths automatically.

The source code for MEME has been available via ftp since June, 1995; however, many potential MEME users do not have the expertise required to download and compile that source code. Others may be unwilling to spend time installing the software before having had a chance to test it. The worldwide web MEME interface was created in order to reach these users.

A major hurdle to creating a web interface to a single, centralized software system is coping with potentially large numbers of simultaneous users. In order to provide efficient access, MEME was ported to the San Diego Supercomputer Center's 400-node Intel Paragon XP/S [SDSC, 1996]. The web interface was linked via a Unix socket directly to the Paragon, allowing a user's click on the "Submit" button to enqueue a job on the supercomputer. For large data sets, the parallelized version of MEME (ParaMEME) runs efficiently on up to 64 nodes in asynchronous SPMD (single program, multiple data) mode.

The ParaMEME project involves two complementary aspects: the parallelization of MEME on a supercomputer, and the development of a web interface to ParaMEME. Accordingly, this paper describes both the parallelization effort and the worldwide web interface.

2 System and methods

MEME consists of approximately 8000 lines of ANSI C source code. MEME was ported to the Paragon using MP++, a message-passing library developed at UCSD [Kohn and Baden, 1996]. The version of MP++ running on the SDSC Paragon is based upon NX, the Paragon's native messaging system; however, a version of MP++ based upon MPI is also available [Fink *et al.*, 1996]. MP++ is portable to a number of platforms, including the CM5, Cray C90 and T3D, PVM, and Sun4, Alpha and RS6000 workstations. On serial workstations, MP++ simulates multiple processors using separate processes. This allows most code development to be done on a workstation and then ported to a parallel computer.

The worldwide web front-end consists of an HTML Forms interface linked to several small Perl scripts and C programs. The client-server socket interface between the web server and the Paragon is also written in C and is linked to NQS queueing software on the Paragon.

3 Algorithm

In order to understand the issues involved in parallelizing MEME, a basic familiarity with the algorithm is essential. A more complete description of the algorithm can be found in [Bailey and Elkan, 1995b]. MEME searches by building statistical models of motifs. Each model consists of a matrix of discrete probability distributions. A value at position (i, j) in the matrix is the probability that letter i will appear in position j of the motif. The zeroth column ($j = 0$) of the matrix models the background, i.e., the probability of letter i in positions outside the motif. There are prior probability distributions on motif models and on motif locations. The overall goal of MEME is to find a model with maximal posterior probability given the data.

A simplified outline of the MEME algorithm is given in Figure 1. The outermost loop (lines 2-15) allows MEME to find multiple motifs in a given data set. After each motif is located, MEME modifies its prior probability distribution on motif locations to approximate erasing of the found motif. This probabilistic erasing prevents MEME from finding the same motif twice.

During each motif search, MEME does a multistart search over the range of possible motif widths (lines 3-12). The algorithm generates a model for each width and then uses a heuristic criterion function $G(\cdot)$ based upon a maximum likelihood ratio test to select the best model.

```
(1) procedure MEME (X : data set of sequences)
(2)   for pass = 1 to num_motifs
(3)     for width = w_min to w_max
(4)       for starting_point = 1 to num_starting_points
(5)         Estimate score of model with this initial model
(6)       end
(7)       Choose initial model with maximum score
(8)       for lambda = lambda_min to lambda_max
(9)         Run expectation-maximization to convergence
(10)      end
(11)      Choose model with maximal G(·)
(12)    end
(13)    Choose and report model with maximal G(·)
(14)    Erase the model from the prior probabilities
(15) end
```

Figure 1: The MEME algorithm.

The search at each possible width has two phases. In the first phase, MEME iterates over a set of possible initial models (lines 4-6). The score assigned to each initial model is its estimated log-likelihood after one expectation-maximization iteration. In the second phase, the algorithm performs expectation-maximization until convergence, using the initial model with the highest score. Without the first phase, the expectation-maximization algorithm usually gets stuck in a local optimum and fails to find the best motif model.

MEME supports three different types of models: OOPS, ZOOPS and TCM. The type of model chosen by a user depends upon prior knowledge concerning the data set. The OOPS model (one occurrence per sequence) assumes that each motif appears exactly once in each sequence in the data set. In the ZOOPS model (zero or one occurrence per sequence), the motif may be absent from some sequences. The TCM model (two-component mixture) is the most general; it allows a motif to appear any number of times in each sequence.

When MEME employs either the ZOOPS or the TCM models, the search for initial models of a given width returns a list. Expectation-maximization is performed on each of these (lines 8-10). Lambda is the mixing parameter of the finite mixture models and is proportional to the number of times the motif occurs in the sequences. The criterion $G(\cdot)$ for choosing an optimal model over various values of lambda is the same one used to select over various motif widths in step (13).

4 Implementation

An ideal program for parallelization contains loops which are "embarrassingly parallel," in which no iteration depends upon the results of the previous iteration. In such a loop, the work from each iteration can be assigned to a separate, parallel processing node. Several aspects of the MEME algorithm are embarrassingly parallel. Of the four main 'for' loops in the pseudocode algorithm above, two are straightforwardly parallelizable, while two are not.

Only in the outermost loop, in which MEME finds successive motifs, does each iteration depend upon its successor. Once a motif is located, it must be subtracted from the prior probability matrix in order to prevent MEME from finding the same motif twice. Thus the search for motif number two cannot begin until motif number one has been found.

Of the other three 'for' loops, the easiest to parallelize is the search for initial models (lines 4-6). MEME finds a set of initial models by performing an independent computation on each subsequence of a given length in the data set. In ParaMEME, each processor independently analyzes a unique subset of the possible initial models, selecting from its share the model with highest score. This selection procedure contains a computationally expensive estimation function involving the initial model and the entire data set. Since ParaMEME runs in SPMD mode, the entire data set is available to every processor. At the end of the loop, all of the processors participate in a maximization reduction, in which the processors cooperatively compare their best initial models, and the highest-scoring model gets propagated to every processor.

The search for initial models parallelizes well in part because the message size is small. A reduction operation across P processors requires sending $O(P)$ messages in time $O(\log P)$, but in this case the message size is only 16 bytes, consisting of the location in the data set of the subsequence that generated the initial model and an associated score. Message latency and throughput are highly variable on the Paragon, so it is difficult to predict communication delays and to choose optimal patterns of communication. Empirically, all messages of 2048 bytes or less face similar delays, of around 2 ms, so 16 byte messages are highly inefficient. For this reason, one complex reduction operation that considers all initial model widths together is performed after the parallelized evaluation of initial model scores, instead of separate reductions for each alternative motif width.

The main expectation-maximization loop (lines 8-10) involves passing larger quantities of data. The maximization reduction based upon $G(\cdot)$ (in line 11) is performed across a set of models. Each model consists of a relatively large matrix of values, containing approximately 22700 bytes of data. According to empirical data from SDSC consultants, the typical communication delay for such a message is around 2 ms also.

We chose not to parallelize the 'for' loop appearing in lines 3-12. In fact, MEME has been optimized so that this loop is incorporated into the two loops just described. Thus, for instance, when ParaMEME divides the possible initial models between processors for evaluation (lines 4-7), each processor actually evaluates its set of initial models at each possible width. It would be possible to parallelize this inner loop and the corresponding inner loop in the expectation-maximization portion of the algorithm. However, the increased parallelism so gained would require extra communication. Because of the expense of communication relative to computation on the Paragon, the increased parallelism would probably not increase efficiency and hence would not improve performance with a fixed number of processors.

Load-balancing

In parallelizing any program, the division of labor between processors must be carried out equitably. If one processor is assigned a particularly difficult portion of the problem, the others will end up sitting idle, causing the overall efficiency of the program to decline. Here, "efficiency" is defined as the real time used by the program running on a single processor divided by the real time when the same program runs in parallel times the total number of processors. In ParaMEME, balancing the expectation-maximization loop turned out to be relatively easy; balancing the search for initial models required considerable effort.

In the expectation-maximization portion of the algorithm, each processor evaluates a set of initial models. These models are stored in an array such that the easiest ones to evaluate — the ones corresponding to the shortest motifs — appear first. Accordingly, processors select initial models in an interleaved fashion, using their unique processor ID numbers to ensure that no initial model gets evaluated by more than one processor. This method ensures that each processor receives an approximately equal share of the work.

A simple, interleaved assignment scheme does not work well in the other loop, the search for initial models. Such models are generated from the sequences in the data set. Each sequence generates a number of initial models proportional to its length. The simplest assignment algorithm doles out one sequence per processor until all the sequences are gone. This plan fares poorly because the number of sequences may be small relative to the number of processors, and because the lengths of the various sequences may vary widely, thereby skewing the workloads of the various processors.

To solve this load-balancing problem, ParaMEME counts the total number of characters in the sequences in the data set and then assigns equal numbers of characters to each processor. Thus, a single processor may search two short sequences and part of a third, while another processor searches a single, long sequence. This algorithm only approximates a perfect load balance, because a small computational overhead is associated with the beginning of each sequence. Thus, a processor which searches a single, 100-character sequence may finish more quickly than a neighboring processor which must search ten 10-character sequences.

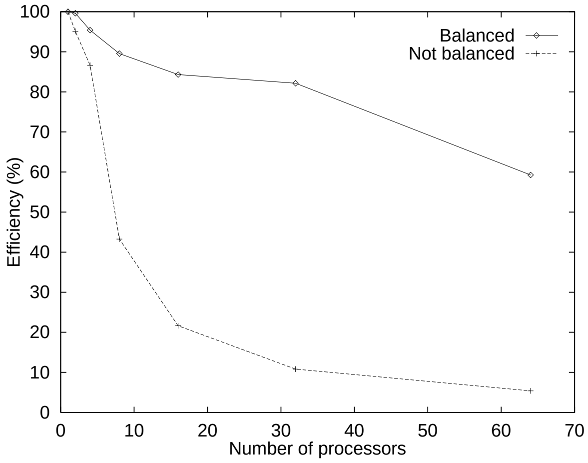

Figure 2: The efficiency of the search for initial models with and without load balancing. The data set consists of four sequences containing a total of 1323 characters. Similar, though slightly less dramatic, results were obtained for larger data sets.

Using appropriate timing data and a more complicated load-balancing algorithm, the search for initial models could be more perfectly balanced; however, even this relatively straightforward balancing scheme improves the efficiency of ParaMEME's search for starting points considerably. Figure 2 shows the efficiency of the search for initial models with and without load balancing. Dividing the work by character rather than by sequence makes a significant difference. For example, using eight processors yields a 90% efficiency in the balanced version, versus a 43% efficiency in the unbalanced version.

Scalability

A key question for any parallel implementation is how well it scales to large numbers of processors. ParaMEME contains two distinct parallelized loops, each of which scales differently. Fortunately, however, the loop that scales more gracefully has higher computational complexity and therefore dominates the running time for large data sets.

The two loops scale differently because the work can be divided more finely in one than in the other. In the first loop — the search for initial models (lines 4-6) — the data set can be divided between any two characters. Thus, the only limitation to scalability is the number of characters in the data set, minus the overhead associated with loading individual sequences into memory and with passing messages between nodes.

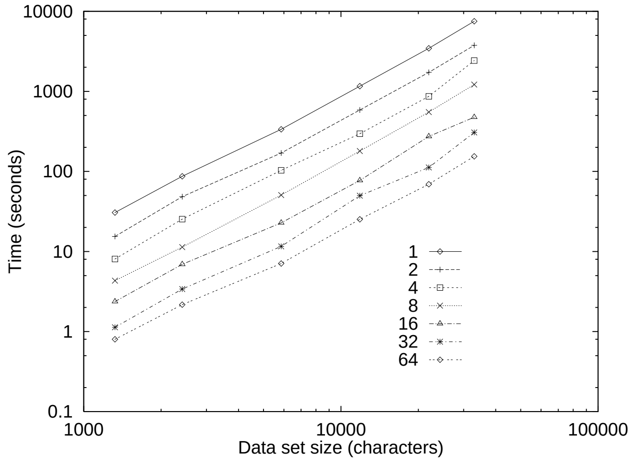

Figure 3: Real time spent in the search for initial models as a function of the size of the data set for different numbers of processors. This data was collected by reporting the time spent in the loop by each node and taking the maximum of those times.

In the expectation-maximization loop (lines 8-10), by contrast, the task can only be easily divided by the number of initial models that were located in the previous loop. The OOPS model performs expectation-maximization on only one initial model per motif width; for the other two models, ZOOPS and TCM, additional initial models are considered in order to allow for varying numbers of motif occurrences per sequence. The number of initial models chosen for the three different models are determined according to the expressions in Table 1. These three expressions put an upper limit on the scalability of the expectation-maximization loop.

model type	number of candidate initial models
OOPS	$2\lfloor\log_2(w/v)\rfloor + 1$
ZOOPS	$(\lfloor\log_2\sqrt{n}\rfloor + 1)(2\lfloor\log_2(w/v)\rfloor + 1)$
TCM	$\lfloor\log_2 N - \frac{1}{2}(\log_2(4\ v\ w))\rfloor(2\lfloor\log_2(w/v)\rfloor + 1)$

Table 1: Number of initial models considered for each model type. v and w are the minimum and maximum motif widths, n is the number of sequences, and N is the total number of characters in the data set.

The equal division of labor in the expectation-maximization loop is further hindered because the expectation-maximization algorithm may converge in different numbers of iterations depending upon the data. Therefore, even when the number of processors is equal to the number of initial models, the expectation-maximization loop may not be balanced perfectly, since some of the processors may finish their work more quickly. Thus we can expect the expectation-maximization loop to scale rather more poorly than the search for initial models.

To test this hypothesis and to judge the relative importance of the scalabilities of the two loops, we ran ParaMEME on six different protein data sets using the ZOOPS model. The data characteristics are summarized in Table 2. These were selected to be typical protein data sets in a range of sizes.

data set	n	min	max	mean	N
PS00904	4	290	377	330.8	1323
PS00043	10	236	254	241.6	2416
PS00098	14	387	547	418.3	5856
PS00402	39	147	514	303.6	11842
PS00180	55	72	729	399.4	21966
PS00228	74	412	457	445.8	32988

Table 2: Characteristics of the data sets. Each data set contains protein sequences from Swissprot [Bairoch, 1994] annotated in the Prosite database [Bairoch, 1992] as true positives or false negatives for the Prosite pattern characterizing a given family. Min, max, and mean refer to the number of residues in individual sequences of the data set, while n and N are the number of sequences and the total number of residues.

Figure 3 shows the real time spent by ParaMEME in the search for initial models as a function of the size of the data set. Each line in the graph represents a different number of processing nodes. Note that the running time continues to improve, even when the number of processors is increased from 32 to 64. The quadratic nature of this phase of ParaMEME is also suggested by the shapes of the curves.

As predicted, the expectation-maximization loop does not scale as gracefully as the search for initial models. Figure 4 shows the real time spent in the second loop as a function of the data set size using the OOPS sequence model. Table 1 indicates that, with the ParaMEME defaults, $v = 12$ and $w = 55$, the OOPS sequence model considers only five initial models. Therefore, the expectation-maximization loop does not finish any faster when the number of processors exceeds five. For the ZOOPS and TCM models, the situation is similar, except that the saturation point occurs at a higher number of processors, since those models involve considering more than five initial models. For example, the ZOOPS and TCM models consider 45 initial models when analyzing the largest (33000-character) data set.

The relatively poor scalability of the second loop is offset by the fact that, for large data sets, the cost of expectation-maximization relative to the cost of the search for initial models decreases. This decrease happens because the expectation-maximization is approximately linear, whereas the search for initial models is quadratic. For small data sets (less than 4000 characters), the two loops require nearly the same amount of computation; however, for larger data sets, the search for initial models become much more expensive. For example, when MEME analyzes a 33000-character data set using the TCM model, it spends almost 97% of the time searching for good initial models.

The overall scalability of ParaMEME is shown in Figure 5. The graph shows the efficiency of ParaMEME as a function of the number of processors for the TCM model; data for the OOPS and ZOOPS models are similar. Each series represents one of the six data sets described above. The graph shows an overall downward trend in efficiency as the number of processors increases. However, for larger data sets, ParaMEME maintains efficiencies over 72% even

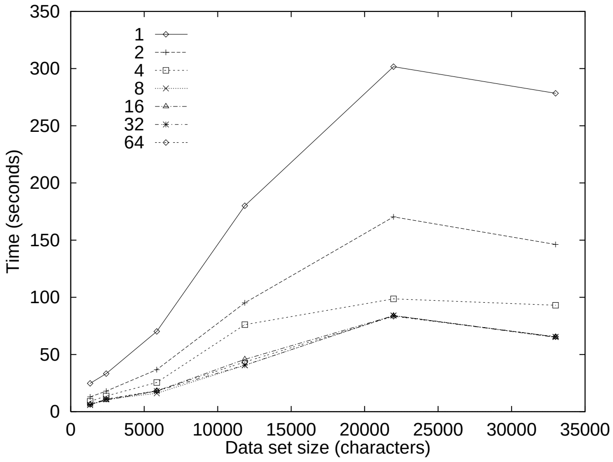

Figure 4: Real time spent in the expectation-maximization loop as a function of the size of the data set for different numbers of processors. This data was collected by reporting the time spent in the loop by each node and taking the maximum of those times.

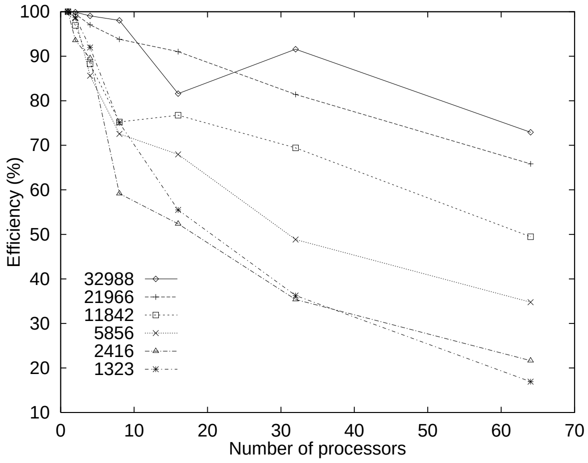

Figure 5: Overall ParaMEME efficiency as a function of the number of processors for data sets of different sizes. This data was collected using the TCM model. Data for the the OOPS and ZOOPS models are similar.

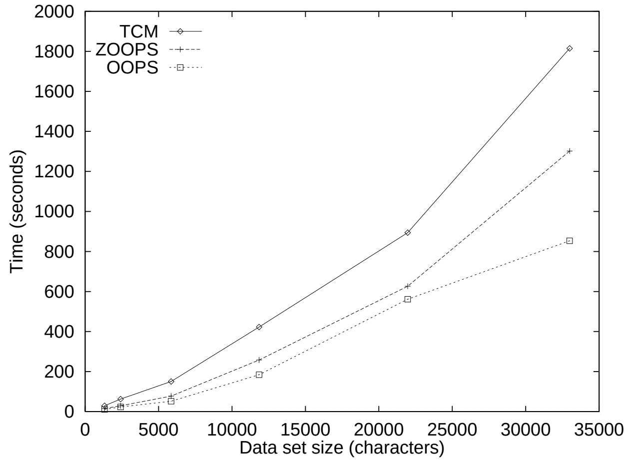

Figure 6: ParaMEME real time as a function of data set size for different types of sequence models. This data was collected using eight nodes on the Paragon, which is the number of nodes assigned to jobs submitted via the ParaMEME web interface.

using 64 processors. Figure 6 shows ParaMEME's real time response as a function of data set size for the three types of sequence models. These data were collected using eight Paragon nodes, so they reflect expected compute times for jobs submitted to the ParaMEME web interface.

Worldwide web server interface

The ParaMEME implementation is efficient, and the Paragon can handle multiple users at once; however, the Paragon itself is not accessible to the typical user. The worldwide web interface was designed to provide easy access to ParaMEME.

To submit data to ParaMEME, a user begins by entering in a web browser an e-mail address, a brief description of the data, and the sequences themselves. The sequences can be in any format supported by the readseq program [Gilbert, 1990], including GenBank, NBRF, FASTA, and many others. The user selects from various ParaMEME options: DNA or protein sequences, the type of model desired (OOPS, ZOOPS or TCM), and the estimated motif width. The default motif width setting is "any," which allows MEME to use statistical modeling techniques to automatically choose the best width of each motif. Next, the user clicks on a "Send" button. The data is immediately parsed, and statistics about the data are displayed on the screen, such as the total number of sequences and average sequence length. Once the user has verified that the data has been properly interpreted, the user clicks on a "Submit" button, and the data is sent to the Paragon. Once ParaMEME is finished, the results are e-mailed to the user.

For each motif MEME discovers, it outputs the subsequences in the input set that match the motif, an information content diagram that visually displays the degree of conservation of each position of the motif, a three-level consensus sequence describing the most conserved residues at each position, and the simplified (rounded to 1 digit) frequency matrix. In addition, several statistical scores are reported for each motif. The most useful such score is probably the

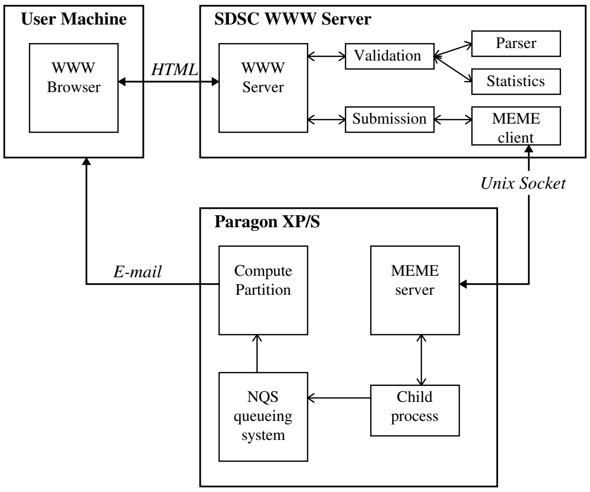

Figure 7: Data flow diagram of the web interface to ParaMEME.

information content (IC). Motifs with IC values of 18 bits or more are usually most useful as probes for searching large sequence databases. Lower IC values tend to give many false positives in searches.

Worldwide web server architecture

Although the submission procedure appears straightforward to the user, a lot goes on behind the scenes. Figure 7 shows the complete architecture of the ParaMEME interface. Three computers are involved: the user's system, the SDSC web server, and the Paragon itself.

One of the goals of the interface is to avoid submitting faulty jobs to the Paragon. Accordingly, as much pre-processing of the data as possible is carried out on the SDSC web server. Once validated, the data is sent through a Unix socket to a server on the Paragon. The server submits the incoming data to a Paragon queue, mails the results back to the user, and then deletes the user's data.

At present, every job submitted to ParaMEME is allocated eight Paragon nodes. Any allocation decision must make a cost-benefit tradeoff between the amount of real time spent solving the problem and the amount of CPU time wasted by inefficient processing. Assigning eight nodes to every job guarantees that, for large jobs, the efficiency will be above 90%, regardless of the model. For smaller jobs, efficiency remains above 58% even for OOPS, the least scalable model. This yields, for a dataset of 10000 characters, a turnaround time of about 200 seconds.

5 Discussion

Any parallel software implementation imposes constraints due to the short lifetime of the hardware. We selected the Paragon because it was a local, available resource which could provide sufficient computational power to handle MEME analyses. However, we selected a general message-passing library so that ParaMEME would not be wedded to a particular hardware platform. For example, ParaMEME will soon be running on the Pittsburgh Supercomputer Center's T3D.

We know of several other computational biology tools running on parallel computers with worldwide web interfaces. The GenQuest Q server at Johns Hopkins University [Jacobson, 1996] provides access to "a specialized parallel computing environment" at the Oak Ridge National Laboratory. Similarly, the BioSCAN server [BioSCAN, 1996] accesses specialized VLSI hardware at the University of North Carolina at Chapel Hill. And a server at the Weizmann Institute of Science provides access to the BIOCCELERATOR [BIOCCELERATOR, 1996] hardware. Other tools, but without web interfaces, include MPSRCH at the Department of Informatics, University of Bergen, Norway [Akselberg and Collins, 1996], which provides an e-mail server to the massively parallel MasPar computer for querying sequence databases using the Smith-Waterman local similarity algorithm, and MultiSearch at the Pittsburgh Supercomputer Center [Kopko *et al.*, 1996], which runs on the Cray T3D and searches sequence databases using a variety of methods, including dynamic programming as well as heuristic methods. Each of these systems uses parallel hardware for the data-intensive task of database searching. The SAM web server at the University of California, Santa Cruz [Hughey and Krogh, 1995], allows the user to build hidden Markov models of families of sequences and to use those models to score sequences and perform multiple alignments. Although SAM has been parallelized for the MasPar computer, the web server is currently running from serial workstations. ParaMEME is therefore unique in providing a web server which uses a parallel computer for a compute-intensive computational biology task, namely discovering motifs.

In conclusion, making MEME easily accessible to biologists who want to analyze their own sequences of nucleic acids and proteins has involved two primary tasks: porting MEME to a parallel supercomputer and building a worldwide web interface to the parallel program. Of these tasks, the first proved to be far more complicated than the second. ParaMEME now runs efficiently on the SDSC Paragon XP/S, and the worldwide web interface has been fully operational since December, 1995. Prospective users are invited to explore the ParaMEME server at the URL `http://www.sdsc.edu/CompSci/Biomed/MEME`.

Acknowledgements:

The authors would like to thank Annalisa Little for important work on the web interface, Joshua Polterock and Kim Claffy for helping with the logistics of the web server, Allan Snavely and George Kremenek for helping set up the Paragon socket interface, Scott Kohn, Stephen Fink and various SDSC consultants (Ken Steube, Nancy Wilkins-Diehr, Richard Frost and others) for their help in debugging the parallel code, Alex Ropelewski for working on the T3D version of ParaMEME, and Michael Baker for acting as a guinea-pig user. Bill Grundy is funded by the National Defense Science and Engineering Grant Fellowship Program. Charles Elkan is funded by a Hellman Faculty Fellowship from UCSD. Paragon time was made available through a grant to NBCR (NIH P41 RR08605), and T3D time was made available by a grant from the Pittsburgh Supercomputing Center through the NIH Cooperative Agreement 1 P41 RR06009.

References

[Akselberg and Collins, 1996] L. Akselberg and J. F. Collins. MPSRCH email server in Bergen. `http://www.ii.uib.no/linda/bio/mpsrch/mpsrch.html`, 1996.

[Bailey and Elkan, 1995a] T. L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using EM. *Machine Learning*, 21(1–2):51–80, October 1995.

[Bailey and Elkan, 1995b] T. L. Bailey and C. Elkan. The value of prior knowledge in discovering motifs with MEME. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 21–29. AAAI Press, 1995.

[Bailey and Elkan, 1996] T. L. Bailey and C. Elkan. The megaprior heuristic for discovering protein sequence patterns. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, page in press. AAAI Press, 1996.

[Bairoch, 1992] A. Bairoch. PROSITE: a dictionary of sites and patterns in proteins. *Nucleic Acids Research*, 20:2013–2018, 1992.

[Bairoch, 1994] A. Bairoch. The SWISS-PROT protein sequence data bank: current status. *Nucleic Acids Research*, 22(17):3578–3580, September 1994.

[BIOCCELERATOR, 1996] BIOCCELERATOR home page. `http://sgbcd.weizmann.ac.il/index.html`, 1996.

[BioSCAN, 1996] The BioSCAN project. `http://www.cs.unc.edu/t̃ell/bioscan/bioscan.html`, 1996.

[D'Esposito *et al.*, 1994] M. D'Esposito, G. Pilia, and D. Schlessinger. BLOCK-based PCR markers to find gene family members in human and comparative genome analysis. *Human Molecular Genetics*, 3(5):735–740, 1994.

[Fink *et al.*, 1996] S. Fink, S. Kohn, and S. Baden. The KeLP programming system. `http://www-cse.ucsd.edu/groups/hpcl/scg/kelp.html`, 1996.

[Gilbert, 1990] D. G. Gilbert. readseq. `gopher://ftp.bio.indiana.edu:70/11/Molecular-Biology/Molbio%20archive/readseq`, 1990.

[Gribskov *et al.*, 1990] M. Gribskov, R. Lüthy, and D. Eisenberg. Profile analysis. *Methods in Enzymology*, 183:146–159, 1990.

[Henikoff *et al.*, 1995] S. Henikoff, J. G. Henikoff, W. J. Alford, and S. Pietrokovski. Automated construction and graphical presentation of protein blocks from unaligned sequences. *Gene-COMBIS, Gene*, 163(GC):17–26, 1995.

[Hughey and Krogh, 1995] R. Hughey and A. Krogh. SAM: Sequence alignment and modeling software system. Technical Report UCSC-CRL-95-7, January 1995.

[Jacobson, 1996] D. Jacobson. GenQuest — the Q server. `http://www.gdb.org/Dan/gq/gq.form.html`, 1996.

[Kohn and Baden, 1996] S. R. Kohn and S. B. Baden. Irregular coarse-grain data parallelism under LPARX. *Journal of Scientific Programming*, 20(3), 1996. To appear.

[Kopko *et al.*, 1996] M. Kopko, A. Ropelewski, and H. Nicholas. Multisearch - parallel sequence database searching program. `http://www.psc.edu/biomed/pages/dissem/multisearch.html`, 1996.

[Lawrence *et al.*, 1993] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, 262(5131):208–214, 1993.

[Schneider *et al.*, 1986] T. D. Schneider, G. D. Stormo, L. Gold, and A. Ehrenfeucht. Information content of binding sites on nucleotide sequences. *Journal of Molecular Biology*, 188:415–431, 1986.

[SDSC, 1996] SDSC's Intel Paragon. `http://www.sdsc.edu/Services/Consult/Paragon/paragon.html`, 1996.

[Tatusov *et al.*, 1994] R. L. Tatusov, S. F. Altschul, and E. V. Koonin. Detection of conserved segments in proteins: iterative scanning of sequence databases with alignment blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 91(25):12091–12095, 1994.

[WSJ, 1996] Internet survey estimates 9.5 million users in U.S. *Wall Street Journal*, page B2, 1996. January 12, 1996.