MP-PIPE: A Massively Parallel Protein-Protein Interaction Prediction Engine *

A. Schoenrock
School of Computer Science
Carleton University
Ottawa, Canada
aschoenr@scs.carleton.ca

F. Dehne
School of Computer Science
Carleton University
Ottawa, Canada
frank@dehne.net

J.R. Green
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
jrgreen@sce.carleton.ca

A. Golshani
Institute for Biochemistry
Carleton University
Ottawa, Canada
agolshan@connect.carleton.ca

S. Pitre
School of Computer Science
Carleton University
Ottawa, Canada
sylverbullit@hotmail.com

ABSTRACT

Interactions among proteins are essential to many biological functions in living cells but experimentally detected interactions represent only a small fraction of the real interaction network. Computational protein interaction prediction methods have become important to augment the experimental methods; in particular sequence based prediction methods that do not require additional data such as homologous sequences or 3D structure information which are often not available. Our *Protein Interaction Prediction Engine* (PIPE) method falls into this category. Park has recently compared PIPE with the other competing methods and concluded that our method "significantly outperforms the others in terms of recall-precision across both the yeast and human data". Here, we present MP-PIPE, a *new* massively parallel PIPE implementation for large scale, high throughput protein interaction prediction. MP-PIPE enabled us to perform the first ever complete scan of the entire *human* protein interaction network; a *massively parallel* computational experiment which took three months of full time 24/7 computation on a dedicated SUN UltraSparc T2+ based cluster with 50 nodes, 800 processor cores and 6,400 hardware supported threads. The implications for the understanding of human cell function will be significant as biologists are starting to analyze the 130,470 *new* protein interactions and possible new pathways in *Human* cells predicted by MP-PIPE.

Categories and Subject Descriptors

J.3 [**Computer Applications**]: Life and Medical Sciences

*Research partially supported by the Natural Sciences and Engineering Research Council of Canada.

ICS'11, May 31–June 4, 2011, Tuscon, Arizona, USA.

General Terms

Parallel Computing

Keywords

Massively Parallel Application, Computational Biology, Protein Interaction Prediction, High Throughput.

1. INTRODUCTION

1.1 Background

Interactions among proteins are essential to many biological functions in living cells but experimentally detected interactions represent only a small fraction of the real interaction networks (e.g. [5]). Computational protein interaction prediction methods have become important to augment the experimental methods (e.g. [8]). Protein-protein interaction (PPI) prediction tools aim to exploit the set of known PPIs, as determined through classical wet-lab techniques, in order to determine whether two proteins will physically interact. There are several approaches to this problem including *sequence-based* prediction techniques (i.e. only the amino acid sequence of the query proteins are required as inputs), examination of the *genetic encoding* of the input proteins, *phylogenetic analysis* of the query proteins, and comparing the query proteins with previously solved *3D structures* of protein complexes (see e.g. [9] for a survey). Sequence based prediction methods are of particular importance in practice because they do not require additional data such as homologous sequences or 3D structure information which are often not available (e.g. [8]). Our algorithm, termed Protein Interaction Prediction Engine (PIPE) [10, 12, 11], falls into this category. An outline of our PIPE algorithm is presented in Section 3. Park [8] has compared PIPE with other sequence based methods [7, 5, 13] and concluded that our method "significantly outperforms the others in terms of recall-precision across both the yeast and human data". Another important consideration is that protein interaction networks are very sparse. Typically less than 0.1% of all possible protein pairs do actually interact. As discussed by Yu et al [14], it is critical that any method that is to be used for protein network wide high-throughput analysis operates

precision =	$\frac{TP}{TP+FP}$	specificity =	$\frac{TN}{TN+FP}$
recall =	$\frac{TP}{TP+FN}$	sensitivity =	$\frac{TP}{TP+FN}$

Table 1: Precision/recall vs. sensitivity/specificity. (TP=number of true positives. FP=number of false positives. TN=number of true negatives. FN=number of false negatives).

at extremely high specificity lest the predicted interactions be completely dominated by false positive predictions. (Biologists often consider sensitivity/specificity instead of precision/recall; see Table 1.1.) At these high specificities (up to 99.95% i.e. less than 0.05 % false positives), our method is particularly effective and achieves significantly higher sensitivity than all other methods [8].

1.2 Summary of Results

Since our PIPE method is more effective and achieves significantly higher sensitivity than competing methods in particular for high specificities (up to 99.95% i.e. less than 0.05 % false positives) [8], PIPE is a prime candidate for scanning the entire protein interaction network (proteome) of organisms. However, this is a massive computational undertaking because, for most organisms, the numbers of proteins pairs is very large. Typical model organisms such as *S.Cerevisiae* and *C.Elegans* have 18,000,000 and 280,000,000 protein pairs, respectively. For the *Human* protein interaction network, 253,000,000 protein pairs need to be tested for possible interactions.

In this paper, we present **MP-PIPE**: a *new* massively parallel PIPE implementation for large scale protein interaction prediction. This is, to our knowledge, the first massively parallel high throughput protein interaction prediction engine which is capable of scanning the entire protein interaction network of organisms. MP-PIPE is able to compute the entire protein interaction network for *C.Elegans* in about one week (on a SUN UltraSparc T2+ based cluster with 50 nodes). This is the first ever complete scan of the *C.Elegans* protein interaction network. For the *Human* protein interaction network, the task was considerably more complicated. Not only does the *Human* protein interaction network have more interactions but the calculation/prediction of these interactions is considerably more time consuming. Whereas nearly all individual interactions for *C.Elegans* could be predicted within seconds, some of the human protein interactions took hours (even days) to predict because of very large numbers of interaction candidate strings. Even though the problem is "embarrassingly parallel", given 253,000,000 protein pairs to work on, the large variation in computation time for individual pairs (from seconds to hours to days) created a massive scale load balancing problem. A considerable portion of our new massively parallel *MP-PIPE* method presented in this paper is dedicated to solving this load balancing problem. In a *large scale* computational experiment, which took three months of full time 24/7 computation on a dedicated SUN UltraSparc T2+ based cluster with 50 nodes, 800 processor cores and 6,400 hardware supported threads, MP-PIPE has been the first system ever to scan the entire *Human* protein interaction network. In addition to the 41,678 previously known *Human* protein interactions, MP-PIPE discovered more than 130,000 *new* protein interactions with high confidence (0.05% false positive rate), potentially more than quadrupling the number of known *Human* protein interactions. The implications for the understanding of human cell function will be significant as biologist are starting to analyze these *new* protein interactions and implied possible new pathways in *human* cells predicted by MP-PIPE.

The remainder of this paper is organized as follows. Section 3 gives a brief review of our sequential PIPE algorithm and Section 4 discusses sequential performance optimization. Section 5 presents MP-PIPE and Section 7 shows a performance evaluation of MP-PIPE on various size clusters. Section 8 outlines the scientific results produced so far by MP-PIPE: the first complete scans of the *C.Elegans* and *Human* protein interaction networks.

2. RELATED WORK

Due to the high computational complexity of many problems in Computational Biology, parallel systems have been designed for numerous bioinformatics tools (see e.g. Rocks Cluster Bio Roll, http://www.rocksclusters.org). The most well known parallel systems for bioinfomatics include parallel simulations such as parallel protein folding [1] and parallel similarity searches such as parallel BLAST (e.g. mpiBLAST [2] and pioBLAST [6]). For protein interaction prediction there is, to our knowledge, no prior work on large scale *parallel* protein interaction prediction systems such as MP-PIPE. As indicated in Section 1.1 there is however a large body of work on sequential protein interaction prediction methods including *sequence-based* prediction techniques, *genetic encoding* based methods, *phylogenetic analysis*, and *3D structure* based techniques. Our Protein Interaction Prediction Engine (PIPE) falls into the category of *sequence based* prediction methods which are of particular importance in practice because they do not require additional data such as homologous sequences or 3D structure information which are often not available. Among the sequence-based methods there are two principle categories: *domain* based methods and *sequence similarity* based methods. Domain based methods (see e.g. [9]) search the query proteins for sequence similarly to known protein domains. If the query proteins contain a pair of domains which have been previously annotated as mediating a PPI, then the query proteins are predicted to also interact. The obvious limitation of domain based methods is that they require previously characterized protein domains for the species in question and cannot identify novel interaction sites outside of the set of known interacting domains. Sequence similarity based methods try to overcome these problems by discovering PPI mediating sequences from known interactions. A number of machine learning approaches (see e.g [5, 7, 13, 14]) examine features derived from the physiochemical properties of the amino acid residues using support vector machines. A limitation of these methods is the complexity of feature extraction, classifier training, and PPI prediction which precludes them from being used for high-throughput protein network wide analysis. More importantly, since protein interactions are typically mediated by small protein segments (15-30 amino acids) and unaffected by the amino acids outside these segments, the prediction accuracy (precision-recall) of these support vector machine based methods is limited because the interaction location is unknown. Our PIPE method [10, 12, 11] overcomes these problems. Rather than using a general purpose learning method such as support vector ma-

chines, PIPE is a custom designed algorithm for detecting interaction sites among proteins. PIPE provides two advantages: improved processing speed and improved prediction accuracy. For protein network wide analysis, involving many many million protein pairs, executing a support vector machine based classification for each pair is not computationally feasible. Most importantly, even it was possible, scanning entire protein networks with machine learning approaches such as [5, 7, 13, 14] would provide results that are completely dominated by false positives. As outlined in Section 1.1, an independent study by Park [8] showed that PIPE significantly outperforms all other sequence based methods in terms of recall-precision. In fact, PIPE is the first method to achieve very high specificities (up to 99.95% i.e. less than 0.05 % false positives) that are sufficient to scan entire protein networks. Furthermore, PIPE requires only positive PPI data for training. Machine learning based methods will change PPI prediction behavior as the ratio of positive-to-negative interactions changes in their training set. Considering that the actual ratio of protein pairs expected to participate in true PPIs is unknown for most species, this is particularly problematic for such methods.

3. REVIEW OF THE BASIC (SEQUENTIAL) PIPE ALGORITHM

For a given organism (e.g. *S.Cerevisiae*, *C.Elegans*, or *Human*) the PIPE algorithm relies on a database of known and experimentally verified protein interactions. For example, for the 22,513 *Human* proteins, only 41,678 interactions are known (out of 253,406,328 possible protein pairs). Considering that experimental verification of a single protein interaction can take days, this is already a massive amount of lab work. Since experimental verification can have large numbers of false positives (up to 40%, see e.g. [10]), the PIPE database is carefully constructed to avoid false data and stores only protein interactions that have been independently verified by multiple experiments. The database represents an *interaction graph* G where every protein corresponds to a vertex in G and every interaction between two proteins X and Y is represented as an edge between X and Y in G. The remainder of this section outlines how, for a given pair (A, B) of query proteins, our PIPE method predicts whether or not A and B interact.

In the first step of the PIPE algorithm, protein A is split up into overlapping fragments of size w by using a sliding window of size w on A. For each fragment a_i of A, where $0 \leq i \leq (|A| - w + 1)$, we search for fragments "similar" to a_i in every protein in graph G. A sliding window of size w is again used on each protein in G, and each of the resulting protein fragments is compared to a_i. For each protein that contains a fragment similar to a_i, all of that protein's neighbors in G are added to an initially empty list, referred to as list R in the remainder.

Before proceeding to the next step, we need to discuss the meaning of the term "similar" used above. To determine whether two protein fragments are similar, a score is generated with the use of a substitution matrix that has a row and column for each amino acid and the value stored at matrix location (i, j) is the probability that the amino acid j is replaced (through evolutionary processes) by amino acid i after a given evolutionary period. The PAM1 matrix [3] represents probabilities of amino acids changing into other amino acids where only a single mutation occurs per 100 amino acids. For PIPE, the PAM120 matrix (i.e. the PAM1 matrix multiplied by itself 120 times) [3] was used to account for longer evolutionary processes. Two sequences of amino acids are considered "similar" if the sum of the PAM120 matrix values of corresponding amino acids pairs is larger than a given threshold S_{PAM}.

In the next step of the PIPE algorithm, protein B is split up into overlapping fragments b_j of size w ($0 \leq j \leq (|B| - w + 1)$) and these fragment are compared to all (size w) fragments of all proteins in the list R produced in the previous step. We then create a *result matrix* of size $n \times m$, where $n = |A|$ and $m = |B|$ and initialize it to contain zeros at the beginning. For a given fragment a_i of A, every time a protein fragment b_j of B is similar to a fragment of a protein Y in R, the cell at position (i, j) in the result matrix is incremented by one. An illustration of the PIPE algorithm is shown in Figure 1.

Row i of the result matrix corresponds to fragment a_i of protein A and column j corresponds to fragment b_j of protein B. The result matrix indicates how many times a pair (a_i, b_j) of fragments co-occurs in protein pairs that are known to interact. A visualization of PIPE's result matrix for two pairs of *S.Cerevisiae* proteins is shown in Figure 2. The x and y axis represent the amino acid fragment locations for proteins A and B, respectively, and the z axis represents the value of the result matrix for each pair of fragment locations. The proteins in Figure 2a are predicted to not interact because the result matrix values are all small. The proteins in Figure 2b are predicted to interact because of the large value (peak) around $x \approx 780$ and $y \approx 150$ which indicates a predicted interaction location.

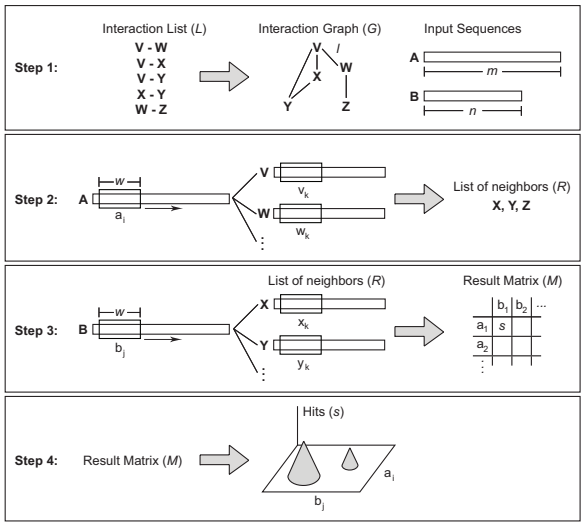

Figure 1: Illustration of the PIPE algorithm.

A possible cause of false positive PIPE predictions are "popular" protein fragments that simply occur very often but have no relationship to protein interactions. It turns out that such false positives typically correspond to very narrow peaks in the result matrix whereas true positives typically correspond to peaks with a wider base. Therefore, a median filter was applied to the output matrix in order to eliminate narrow peaks [12]. However, applying a median filter is com-

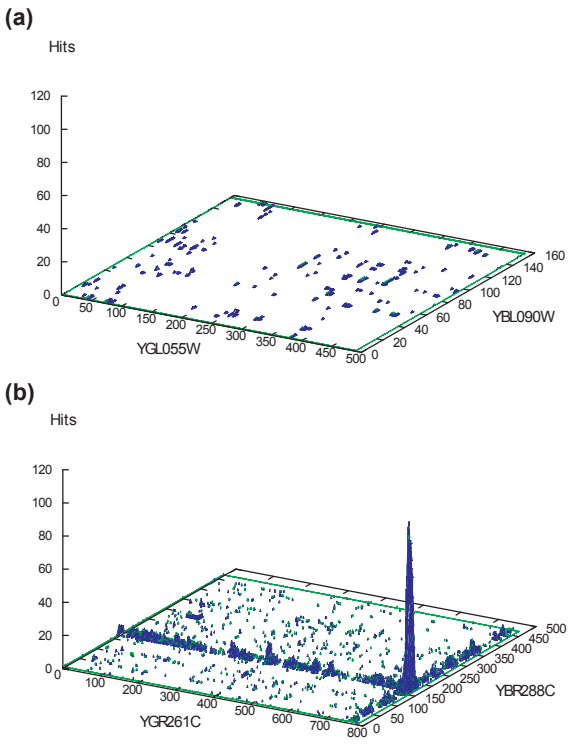

Figure 2: Visualization of PIPE's result matrix for two pairs of *S.Cerevisiae* proteins.

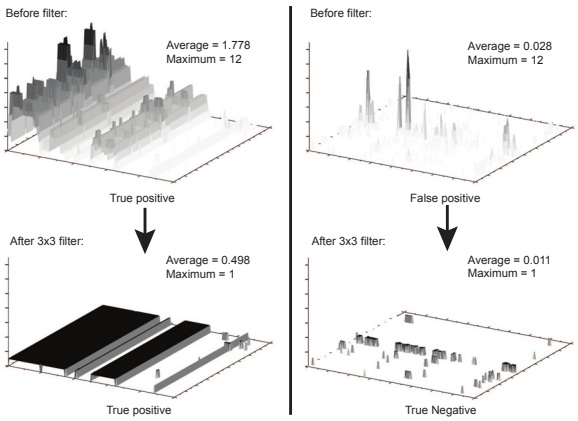

Figure 3: Illustration of the simplified median filter. Impact of applying a 3x3 filter on a PIPE result matrix. Left: Example of two interacting proteins (true positive). Right: Example of two not interacting proteins (true negative).

putationally expensive. Thus, a simplified median filter was used: For a given cell c, if its neighbors consisted of more zeros than non-zeros then c would be set to zero, otherwise c would be set to 1. After this simplified median filter, the average value of all cells of the result matrix is calculated, and if the average is above a given threshold then the proteins are predicted to interact. An illustration is shown in Figure 3. The threshold parameters of PIPE were tuned using a true positive set and true negative set of 1,274 pairs each and applying leave-one-out cross-validation.

4. SEQUENTIAL PERFORMANCE OPTIMIZATION

It is important to note that before proceeding with a parallelization of PIPE, considerable efforts were made to *optimize* the sequential PIPE implementation. We would like to highlight three performance improvements that were particularly successful. (1) The character based amino acid representation of the proteins was converted into binary. This rather simple change removed the need for a character-to-index lookup when adding up the PAM120 scores. (2) The "sliding the window" process across proteins was improved, making use of incremental updates (moving both fragments one position in sync requires only one addition and deletion each). (3) We observed that many protein fragment comparisons are repeated multiple times, in particular when predicting interactions between many protein pairs. After all, query proteins and their fragments are from the given protein set of the organism under consideration. Therefore, we pre-computed all possible protein fragment comparisons and stored all matches of similar fragments. For a set of query proteins, all relevant fragment comparisons are found via lookup instead of string comparison. In the remainder, we will refer to the graph G of previously known protein interactions as the *PIPE interaction graph* and the above precomputed fragment comparisons as the *PIPE database*.

5. MP-PIPE OVERVIEW

An important design goal for MP-PIPE was to obtain a *flexible* and *portable* parallel system that can be scaled to parallel architectures of different size depending on the complexity and size of the protein interaction network to be processed. We targeted in particular three popular parallel architectures: small scale local workstation networks, medium scale processor clusters, and large scale processor clusters. As discussed in Section 1, the main goal for MP-PIPE is to enable a *complete scan of all protein pairs for a given organism*, and the most important application is the first ever scan of the entire *Human* protein interaction network with its 253,000,000 protein pairs of which only a fraction has been evaluated so far. Another important requirement for MP-PIPE was therefore *fault tolerance* and *crash recovery*. The MP-PIPE run for the *Human* proteome took three months of full time 24/7 computation on a dedicated SUN UltraSparc T2+ based cluster with 50 nodes, 800 processor cores and 6,400 hardware supported threads. Needless to say, there were many hardware and system software crashes during such an extended time period where the cluster was run at maximum capacity.

In addition, preliminary experiments showed an interesting challenge that made the scan of the *Human* protein interaction network considerably harder than scanning the proteomes of other organisms such as *S.Cerevisiae* and *C.Elegans*. Note that the number of *Human* proteins and protein pairs is not exceptional. Simple organisms such a *C.Elegans* actually have more proteins and protein pairs than *Human*. However, the *Human* protein interaction network has more interactions and a more complex structure. In particular, the calculation/prediction of these interactions is consid-

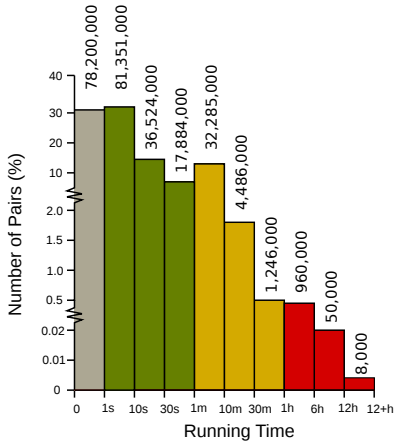

Figure 4: Distribution of running times for *Human* protein-protein interaction prediction. (Numbers above bars indicate approximate number of protein pairs with a running time within the given range.)

erably more time consuming. Previous PIPE experiments for *S.Cerevisiae* [10, 12, 11] and experiments for *C.Elegans* reported in Section 8 below showed that PIPE can process each individual protein pair within seconds. However, for *Human* proteins, the picture changes dramatically. As shown in Figure 4, the running time for one individual protein pair can fluctuate between less than a second and more than 12 hours. *Human* proteins have a much more complex structure which appears to lead, in some cases, to a very large number of fragment similarities found by PIPE. For two query proteins A and B, if a fragment a_i in A finds many nodes in G associated with proteins that contain fragments similar to a_i then every fragment b_j in B needs to be tested against all those proteins. If that happens for many a_i, as seems to be the case for certain *Human* proteins, then the running time can escalate from seconds to hours. Furthermore, a large number of fragment matches and excessive runtime does not increase the likelihood that those proteins are predicted to interact. One could imagine that many fragment matches imply high peaks in the result matrix and one could maybe simply stop the computation after a few minutes and predict an interaction but this is not the case. Many fragment matches do not necessarily lead to high counts in the result matrix. In fact, our experiments show that protein pairs with many fragment matches and requiring a large computation time have nearly the same probability of being predicted to interact than those protein pairs than can be processed within seconds. Therefore, even though processing 253,000,000 protein pairs is essentially an embarrassingly parallel problem, those 1,000,000 protein pairs that require more than one hour of processing time and in particular those 8,000 protein pairs that require more than 12 hours of processing time (see Figure 4) create an interesting load balancing problem.

The basic structure of MP-PIPE is a two-level master/slave model. A single *MP-PIPE scheduler* process is in charge of managing the main list of protein pairs to be processed as well as reporting the results. The MP-PIPE scheduler distributes work to several *MP-PIPE worker* processes in packets. Each packet contains a relatively small number of protein pairs. Each MP-PIPE worker executes the PIPE algorithm on protein pairs received from the MP-PIPE Scheduler. By giving each worker only a relatively small amount of work at a time we ensure that if a worker does get stuck with an abnormally hard packet (one or more of those very time consuming protein pairs), the other workers will continue to work on their packets and, when they finish, they will request more work from the scheduler process and continue to work. It should be noted however that if the packet size is too small then the amount of communication between the scheduler and worker processes will negatively impact the running time of the system. It is therefore important to balance the packet size between being too small (too much communication overhead) and too large (too much work imbalance).

Algorithm 1: MP-PIPE Scheduler.

```
Split protein pairs into packets.
while packets remain do
    receive work request from worker x
    receive previous results from worker x
    send packet to worker x
    write results to output file
foreach worker process do
    receive work request from worker x
    receive previous results from worker x
    send KILL_SIGNAL to worker x
    write results to output file
```

Algorithm 2: MP-PIPE Worker.

```
Load PIPE interaction graph
current_packet ← ∅
current_results ← ∅
work_available ← TRUE
foreach thread in parallel do
    while work_available do
        if current_packet = ∅ then
            request work from scheduler
            send current_results to scheduler
            receive message from scheduler
            if message = KILL_SIGNAL then
                work_available ← FALSE
                BREAK
            else
                current_packet ← message
        retrieve pair from current_packet
        run PIPE algorithm on pair
        add results to current_results
```

To improve load balancing, MP-PIPE uses a two-level model where each MP-PIPE worker consists again of a number of parallel threads, called *worker threads*, among whom it distributes the protein pairs to be processed. The worker threads of an MP-PIPE worker are envisioned to be executed on a shared memory multi-core processor. A major concern is the efficient use of memory. The PIPE interaction graph

G and the large database of pre-computed protein fragment similarity matches requires considerable amounts of memory. For MP-PIPE, the PIPE interaction graph stored at an MP-PIPE worker was re-designed to become a parallel data structure on which all worker threads for that worker can operate concurrently. Much care was taken to implement the PIPE interaction graph and database as memory efficient as possible so that a single shared copy fits into the main memory of a processor node executing an MP-PIPE worker. In addition, the pre-computed database files were not all loaded at once at the start of the computation but were loaded only when needed. This allowed more threads to run simultaneously on a given processor node by reducing the overall memory usage.

The scheduler/worker part of MP-PIPE was implemented using MPI (on the SUN T2+ cluster: SunMPI within Sun Cluster Tool 6) and the worker threads within each MP-PIPE worker were implemented in OpenMP (on the SUN T2+ cluster: OpenMP within the SunOS SPARC 5.9C compiler). Pseudo code for the MP-PIPE scheduler and MP-PIPE workers are shown in Algorithm 1 and Algorithm 2, respectively.

Once the interaction graph G has been loaded by a MP-PIPE worker, it splits into a user defined number of worker threads. Each thread first checks if the packet they are working on still has pairs to process. If not, the thread requests more work from the scheduler process. When the scheduler process responds, the thread checks if the message is a signal to stop. If it is, it sets the *work_available* flag and exits. If not, it sets up the incoming packet to be processed by itself and the other threads, and continues. While there is work available in the current packet, each thread simply takes a pair from the packet, runs the PIPE algorithm on that pair (using the shared PIPE interaction graph) and then adds the result to the results array. It is important that the first thread to notice that the current packet is empty communicates directly with the scheduler and requests a new packet for the entire group of threads of the MP-PIPE worker without any interruption of the other threads. We refer to this mechanism as the *dynamic work request* design. The alternative *static work request* design is discussed below in Section 6.

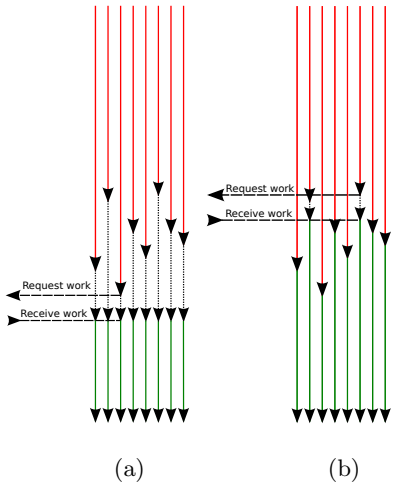

Figure 5: MP-PIPE worker thread implementation alternatives. The arrows represent worker threads where the red lines represent work on the first packet and the green lines represent work on a new packet after the first packet has been completed. (a) Static work requests. (b) Dynamic work requests implemented for MP-PIPE.

6. DISCUSSION OF DESIGN ALTERNATIVES

Before arriving at the MP-PIPE algorithm outlined above, we considered various design alternatives, some of which we discuss here. One possible alternative solution would have been to parallelize the PIPE execution for each individual protein pair. As outlined in Section 3, each PIPE execution consists of a number of graph and string searches. Parallelizing graph and string searches is a non-trivial task and is known to lead in most cases to less than optimal speedup in practice (e.g. [4]). Furthermore, it would have interfered with some of the optimizations outlined in Section 4. For example, the sliding window optimization discussed in Section 4, Item (2), makes use of incremental updates to improve speed but introduces sequential dependencies between window queries. In contrast, MP-PIPE makes use of the massive parallelism available due to the millions of protein pairs that require separate PIPE executions. There are however various alternatives possible with respect to MP-PIPE's two level master slave structure using MPI and OpenMP. For example, we could have chosen an MPI only based implementation where each processor node as well as all threads within each node are MPI based. However, MPI threads do not support shared data structures and since each thread requires access to the PIPE interaction graph, we would have needed several copies of the interaction graph on each node. The memory available would have severely restricted the number of threads that can be executed on each node. For example, for the SUN UltraSparc T2+ based cluster with 50 nodes used in our experiments, it would not have been possible to utilize the available 6,400 hardware supported threads.

The MP-PIPE design outlined in Section 3 was aimed at providing high efficiency for a wide range of parallel architectures including small scale workstation clusters and very large scale clusters with large numbers of hardware supported threads. MP-PIPE's two level master slave structure also has the advantage to require only very little inter-node communication. In fact, after the PIPE interaction graph has been loaded onto each node, the only communication required are work packets and results sent between the scheduler and the workers. The amount of communication needed is in fact so low that we decided to have only one single scheduler process on one node responsible for all workers on all nodes. We also considered double buffering such as overlapping receiving results from workers and sending packets to workers in Algorithm 1. However, since the amount of communication between scheduler and workers as well as the computational load on the scheduler are so small, this had no measurable impact on MP-PIPE's performance.

Another important design choice is the *dynamic work request* scheme for assigning work to the worker threads of each node discussed in Section 5. While there is work available in the current packet, each thread takes a protein pair from the packet, runs the PIPE algorithm on that pair (using

the shared PIPE interaction graph) and then adds the result to the results array. It is important that the first thread to notice that the current packet is empty communicates immediately with the scheduler and requests a new packet for the entire group of threads of the MP-PIPE worker without any interruption of the other threads. We refer to this as the *dynamic work request* design. Its implementation in OpenMP is not exactly in the spirit of OpenMP program design but the *dynamic work request* design is very important for efficiency because of those protein pairs with extremely long processing times. As illustrated in Figure 5, the alternative *static work request* design where all worker threads need to join/sync first before a new package of protein pairs is requested from the scheduler leads to wait times and is less efficient. In fact, because of those protein pairs with extremely long processing times, the difference in performance between the static and dynamic work request designs can be very large.

7. MP-PIPE PERFORMANCE

MP-PIPE's performance was tested on a small cluster (Cluster 1), a medium size cluster (Cluster 2) and a large cluster (Cluster 3). The precise cluster configurations are outlined below. The benchmark contains two tests for each cluster. The first test is designed to evaluate how MP-PIPE scales as more threads are used by a worker process. This test is done by using a single worker on a single cluster node, first starting with one thread and then increasing the number of threads until the point of maximum performance is found. For the second test, we evaluate how MP-PIPE scales as more workers are added, using the optimal number of threads per worker found in the first test. Tests are performed using a set of 5,000 random protein pairs except for some of the tests on the large cluster (Cluster 3) which required larger data sets (50,000 and 500,000 random protein pairs). All reported running times (and resulting speedups) are averages of 100 experiments and measured as wall clock times (between start of program until termination of the last cluster node).

7.1 Small Cluster

We tested MP-PIPE on a small cluster (Cluster 1) with six nodes connected by a gigabit Ethernet switch. Each node consisted of an Intel quad core processor (1.6GHz), 8 GB DDR2 RAM and a 320 GB hard drive, running Ubuntu Linux (10.04). The first test was performed to determine the optimal number of threads for each worker as described above. The results displayed in Figure 6 show the average running times and speedups with different numbers of threads for one worker running on one processor node. The data sets consisted of 5,000 random protein pairs. As shown in Figure 6, 5 threads per worker process provides the best speedup of 3.122. The second test examined how MP-PIPE scales when more processor nodes are added. Each processor node ran one MP-PIPE worker with 5 worker threads. The result are shown in Figure 7. Interestingly, we obtain a slightly above linear speedup with respect to the number of workers (processor nodes). The effect is due to the increase in total cache size and the fact that the MP-PIPE scheduler runs on one of the nodes together with one of the MP-PIPE workers. The scheduler process requires only a very small fraction of the computation required by an MP-PIPE worker but contributes to the artifact of a slightly above linear speedup. The total speedup obtained by MP-PIPE on Cluster 1 is 21.485, the product of the speedup obtained by each worker/node (3.122) and the speedup for 6 workers (6.882).

No. Threads per Worker	Average Running Time (s)	Speedup
1	2318.547131	1
2	1213.482118	1.911
3	835.571807	2.775
4	777.797144	2.981
5	742.669497	3.122
6	746.089079	3.108
7	745.651284	3.109
8	775.769684	2.989
9	790.394030	2.933
10	804.553819	2.882

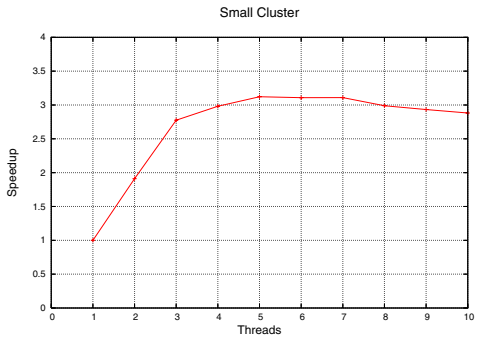

Figure 6: MP-PIPE performance for different numbers of threads per worker on a small cluster (Cluster 1). Average running times for 5,000 random protein pairs, using one worker.

7.2 Medium Size Cluster

We tested MP-PIPE on a medium size cluster (Cluster 2) with 32 nodes connected by a gigabit Ethernet switch. Each node consisted of four Opteron Cores (2.2 GHz), with 8 GB RAM running Linux (ROCKS). Again, the first test was performed to determine the optimal number of threads for each worker as described above. The results are displayed in Figure 8 and show the average running times and speedups with different numbers of threads for one worker running on one processor node. The data sets consisted of 5,000 random protein pairs. As shown in Figure 8, 5 threads per worker process provides the best speedup of 2.764. The second test examined again how MP-PIPE scales when more processor nodes are added. Each processor node ran one MP-PIPE worker with 5 worker threads. The results are shown in Figure 9. Interestingly, we obtain again a slightly above linear speedup with respect to the number of workers (processor nodes). As discussed in Section 7.1, the effect is due to the increase in total cache size and the fact that the MP-PIPE scheduler runs on one of the nodes together with one of the MP-PIPE workers. The total speedup obtained by MP-PIPE on Cluster 2 is 107.807, the product of the speedup obtained by each worker/node (2.764) and the speedup for 32 workers/nodes (39.004).

Number of Workers	Average Running Time (s)	Speedup
1	742.669497	1
2	331.018381	2.244
3	232.867967	3.189
4	168.935589	4.396
5	131.902550	5.630
6	107.919543	6.882

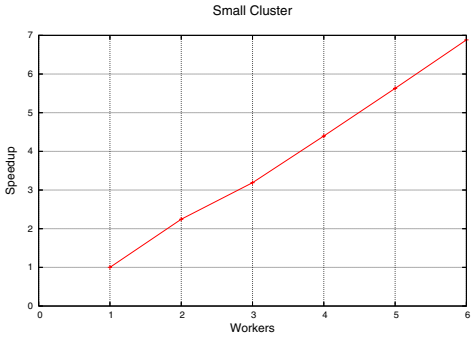

Figure 7: MP-PIPE performance for different numbers of workers on a small cluster (Cluster 1). Average running times for 5,000 random protein pairs, using 5 threads per worker.

No. Threads per Worker	Average Running Time (s)	Speedup
1	3750.0589016	1
2	2027.0536836	1.850
3	1423.47368383	2.634
4	1359.32627749	2.759
5	1356.51471854	2.764
6	1372.91203179	2.731
7	1368.82533481	2.740
8	1406.15620157	2.667
9	1427.07301278	2.628
10	1441.51278317	2.601

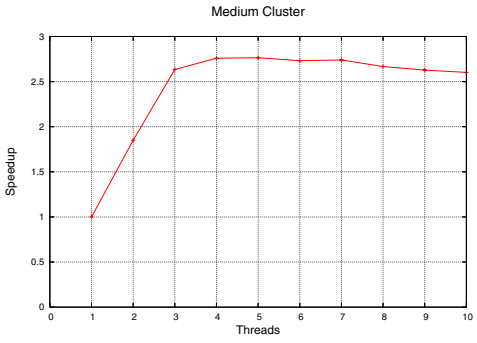

Figure 8: MP-PIPE performance for different numbers of threads per worker on a medium size cluster (Cluster 2). Average running times for 5,000 random protein pairs, using one worker.

Number of Workers	Average Running Time (s)	Speedup
1	1356.51471854	1
2	574.4489908	2.361
4	283.32493562	4.788
8	145.42208586	9.328
16	68.92684697	19.680
32	34.77818442	39.004

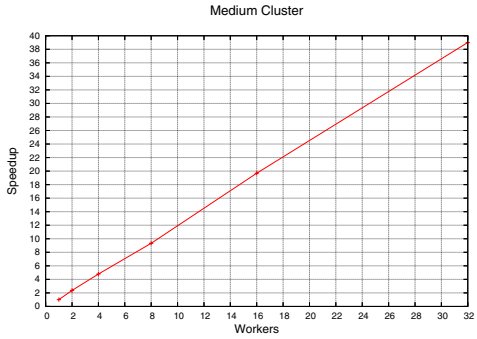

Figure 9: MP-PIPE performance for different numbers of workers on a medium size cluster (Cluster 2). Average running times for 5,000 random protein pairs, using 5 threads per worker.

7.3 Large Cluster

We tested MP-PIPE on a UltraSparc T2+ based "Victoria Falls" cluster (Cluster 3) with 50 nodes. Each node contains 2 UltraSparc T2+ chips (1.2 Ghz) with 8 compute cores per chip and 32 GB RAM. Each of these compute cores has 8 hardware supported threads, giving each node 128 hardware supported threads and providing 6,400 hardware supported threads in total.

The first test was again performed to determine the optimal number of threads for each worker process (running on one node). Due to the large number of hardware supported threads on a Victoria Falls node, the 5,000 random protein pairs were not sufficient to show the full scaling capabilities of a node. For this reason a second data set comprised of 50,000 random protein pairs was created. The 5,000 pair data set was used to examine the scaling of the code with a small number of threads (1 – 16 threads) and then the 50,000 pair data set was used for tests with 16 or more threads. The 50,000 pair data set was not used on the smaller number of threads because of the amount of time it would take to process. To calculate the speedup on the higher number of threads processing the 50,000 pair data set, the running time of a single thread to processes the 50,000 pair data set was approximated by taking the speedup from the trials on the 5,000 pair data set. The results are shown in Figure 10. The best result was achieved for 512 threads leading to a speedup of 8.701. The speedup curve shown in Figure 10 is essentially flat for more than 128 threads and we found that using more than 512 threads creates memory problems. In total, MP-PIPE's worker threads makes reasonably efficient use of the UltraSparc T2+ architecture. The slightly lower speedup compared to the small and medium size clusters can be explained by the increased number of threads spending more time waiting for new packets. On the small and medium size clusters, each node has only 5 concurrent threads and

No. Threads per Worker	Average Running Time (s)	Speedup
(5,000 protein pairs)		
1	9308.75325356	1
2	6622.24860655	1.406
4	4438.87609902	2.097
8	3259.65337563	2.856
16	2410.89934953	3.861
(50,000 protein pairs)		
1	80408.797712014086	1
16	20825.2934437	3.861
32	15325.5586534	5.247
64	11530.1009018	6.974
128	9259.13093131	8.684
256	9302.602417	8.644
512	9241.05782809	8.701

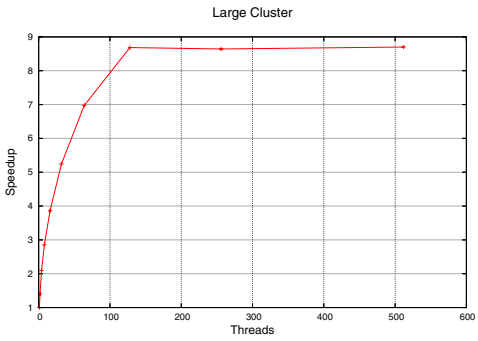

Figure 10: MP-PIPE performance for different numbers of threads per worker on a large cluster (Cluster 3). Average running times for 50,000 random protein pairs, using one worker.

the chance of a thread waiting for another thread to finish communicating with the scheduler (for a new work packet) is very small. On the Victoria Falls cluster, each worker has 512 threads which increases the chance that threads have to wait. This also highlights the importance of the dynamic work request mechanism implemented for MP-PIPE (see Section 5).

The second test examined again how MP-PIPE scales when more workers/processors are added. Each cluster node ran one MP-PIPE worker with 512 worker threads. The result are shown in Figure 11. The performance of MP-PIPE scales almost linearly as the number of nodes used increases. This is an excellent result considering the vast number of threads involved.

8. SCIENTIFIC RESULTS

8.1 First Ever Complete Scan Of The C. Elegans Proteome

The first demonstration of MP-PIPE's performance was the first ever scan of the entire *C. Elegans* proteome.

- Total number of *C. Elegans* proteins: 23,684
- Total number of protein pairs to examine: 280,454,086
- Total number of known protein interactions: 6,607

Number of Workers	Average Running Time (s)	Speedup
1	18244.3975384	1
2	8992.99307318	2.029
4	4571.25758775	3.991
8	2294.45482244	7.952
16	1183.31196108	15.418
32	620.47111997	29.404

Figure 11: MP-PIPE performance for different numbers of workers on a large cluster (Cluster 3). Average running times for 500,000 random protein pairs, using 512 threads per worker.

- Total number of proteins with at least one known interacting partner: 3,460
- Total number of proteins with no known interacting partners: 20,224
- Largest number of known interactions partners for a single protein: 512
- Smallest number of known interactions partners for a single protein: 0
- Average number of known interactions per protein: 0.55
- Average number of known interactions per protein with at least one interaction: 3.82

MP-PIPE evaluated all 280,454,086 possible protein pairs in the *C. Elegans* proteome. The work was split between the Victoria Falls cluster (Section 7.3) and Cluster 2 (Section 7.2). That is, MP-PIPE was run concurrently on both clusters. On the Victoria Falls cluster, 50 nodes were used. Each node executed one worker process running 512 worker threads. Hence, a total of 25,600 parallel computational threads were running on 6,400 hardware supported threads. On Cluster 2, 60 nodes were used each with its own worker process running 5 threads. This represents 300 parallel computational threads running on 240 cores. Both clusters executed sequences of "jobs" where each job contained approximately 10% of the total set of protein pairs to be processed. After a week of 24/7 computation, the first ever scan of the entire *C. Elegans* proteome was completed. At a specificity of 99.99%, MP-PIPE predicted 37,572 protein interactions. Of these high confidence predictions (0.001% false positive rate), 31,065 protein interactions are novel. Given that only

6,607 protein interactions are known for *C. Elegans*, this greatly increases our knowledge of the *C. Elegans* proteome.

Besides experimental verification and leave one out cross-validation, another standard method for evaluating a protein interaction prediction method is the "co-location test" which checks whether the proteins pairs predicted to interact are located in the same cellular component, have the same molecular function, or are involved in the same biological process (e.g. [8]). The results are shown in Figure 12. The percentage of pairs predicted by MP-PIPE that have similar function, occur in the same cellular component and participate in the same cellular process is 1.3%, which is consistent with the percentage for previously reported protein pairs (1.9% for 6,607 pairs). In contrast, for randomly selected protein pairs, the percentage of pairs that have similar function, occur in the same cellular component and participate in the same cellular process is only 0.2%. Similarly, for molecular function and biological processes, the MP-PIPE results are very similar to the previous experimentally confirmed protein interactions and very different from a control group of random protein pairs. It is important to note that the PIPE algorithm has no knowledge of the molecular function of proteins or which biological processes they are involved in. The probability that MP-PIPE could have found such co-occurrences by chance is extremely small.

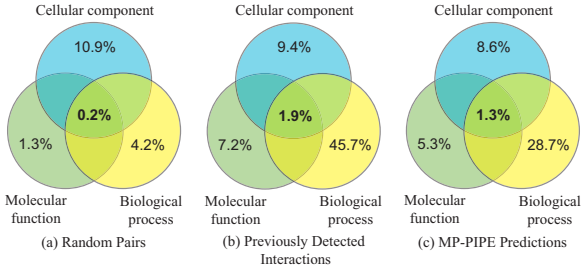

Figure 12: Co-Location Test: Percentages of *C. Elegans* protein pairs located in the same cellular component, with the same molecular function, or involved in the same biological process. (a) Random protein pairs [control group]. (b) Previous experimentally confirmed protein interactions. (c) Protein interactions predicted by MP-PIPE.

8.2 First Ever Complete Scan Of The Human Proteome

After the successful scan of the *C. Elegans* proteome, our group started a very large MP-PIPE run to perform the first ever complete scan of the *Human* proteome.

- Total number of *Human* proteins: 22,513
- Total number of protein pairs to examine: 253,406,328
- Total number of known protein interactions: 41,678
- Total number of proteins with at least one known interacting partner: 9,459
- Total number of proteins with no known interacting partners: 13,054

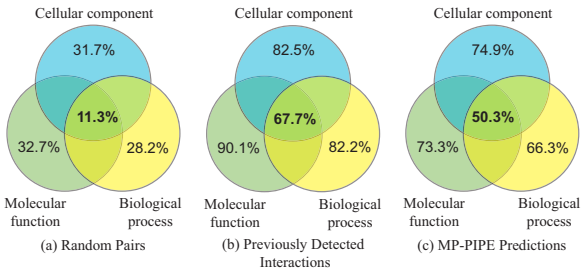

Figure 13: Co-Location Test: Percentages of *Human* protein pairs located in the same cellular component, with the same molecular function, or involved in the same biological process. (a) Random protein pairs [control group]. (b) Previous experimentally confirmed protein interactions. (c) Protein interactions predicted by MP-PIPE.

- Largest number of known interactions partners for a single protein: 265
- Smallest number of known interactions partners for a single protein: 0
- Average number of known interactions per protein: 3.70
- Average number of known interactions per protein with at least one interaction: 8.81

MP-PIPE evaluated all 253,406,328 possible protein pairs in the *Human* proteome. The work was again split between the Victoria Falls cluster (Section 7.3) and Cluster 2 (Section 7.2). The *Human* proteome has almost 7 times more known interactions than the *C. Elegans* proteome. The average *Human* protein has more than double the known interactions than a *C. Elegans* protein. Coupled with the fact that the *Human* proteins are, on average, longer than the *C. Elegans* proteins, this increases the complexity of scanning the entire *Human* significantly. Furthermore, as outlined at the beginning of Section 5 and illustrated in Figure 4, the running time for one individual protein pair can fluctuate between less than a second and more than 12 hours. This creates an additional load balancing problem that has been discussed in detail in Section 5. In fact, some individual protein pairs required 6 days of computation.

On the Victoria Falls cluster (Section 7.3), 50 nodes were used each with their own MP-PIPE worker process running 256 threads. This implies 12,800 parallel computational threads running on 6,400 hardware supported threads. The number of threads per node was scaled down from 512 threads used in the *C.Elegans* scan due to the fact that each individual thread needed significantly more memory. The Victoria Falls cluster was used to process the vast majority of protein pairs. If one of its worker threads got stuck with a protein pair that was running more than 12 hours, that protein pair was off-loaded to Cluster 2 (Section 7.2) since its individual cores are much more powerful than a single Victoria Falls thread.

After 3 months of 24/7 computation on the 50 fully dedicated nodes of the Victoria Falls cluster (plus the additional computation on Cluster 2), MP-PIPE finished the first ever

complete scan of the *Human* proteome. At a specificity of 99.95%, MP-PIPE predicted 172,183 protein interactions. Of these high confidence predictions (0.05% false positive rate), 132,710 protein interactions are novel. Given that 41,678 *Human* protein interactions are known, MP-PIPE potentially more than quadrupled our knowledge of the *Human* proteome.

Besides experimental verification and leave one out cross-validation, another standard method for evaluating a protein interaction prediction method is the "co-location test" which checks whether the proteins pairs predicted to interact are located in the same cellular component, have the same molecular function, or are involved in the same biological process (e.g. [8]). The results are shown in Figure 13. The percentage of pairs predicted by MP-PIPE that have similar function, occur in the same cellular component and participate in the same cellular process is 50.3%, which is consistent with the percentage for previously reported protein pairs (67.7%). In contrast, for randomly selected protein pairs, the percentage of pairs that have similar function, occur in the same cellular component and participate in the same cellular process is only 11.3%. Similarly, for molecular function and biological processes, the MP-PIPE results are very similar to the previous experimentally confirmed protein interactions and very different from a control group of random protein pairs. As previously stated, the PIPE algorithm has no knowledge of the molecular function of proteins or which biological processes they are involved in and the probability that MP-PIPE could have found such co-occurrences by chance is extremely small.

9. CONCLUSION

In this paper, we presented **MP-PIPE**: a *new* massively parallel PIPE implementation for large scale protein interaction prediction. In a *large scale* computational experiment, which took three months of full time 24/7 computation on a dedicated SUN UltraSparc T2+ based cluster with 50 nodes, 800 processor cores and 6,400 hardware supported threads, MP-PIPE has been the first system ever to scan the entire *Human* protein interaction network (253,406,328 protein pairs). The biggest challenge here was that, while most protein pairs can be processed within seconds or minutes, some protein pairs require more than 12 hour or even several days of computation. This creates a non-trivial load balancing problem which MP-PIPE has been able to overcome. At a specificity of 99.95%, MP-PIPE predicted 172,183 protein interactions. Of these high confidence predictions (0.05% false positive rate), 132,710 protein interactions are novel. Given that currently only 41,678 *Human* protein interactions are known, MP-PIPE potentially more than quadrupled our knowledge of the *Human* proteome. We are currently building a publicly accessible database with the data generated by MP-PIPE. The implications for the understanding of human cell function will be significant as biologist are starting to analyze these *new* protein interactions and implied possible new pathways in *human* cells predicted by MP-PIPE.

10. REFERENCES

[1] A. Beberg and V. S. Pande. Folding@home: lessons from eight years of distributed computing. In *IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8, 2009.

[2] A. E. Darling, L. Carey, and W. Feng. The design, implementation, and evaluation of mpiblast. In *ClusterWorld*, 2003.

[3] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, pages 345–352+. 1978.

[4] F.Dehne, A.Ferreira, E.Caceres, S.Song, and A.Roncato. Efficient parallel graph algorithms for coarse grained multicomputers and bsp. *Algorithmica*, 33:2:183–200, 2002.

[5] Y. Guo, L. Yu, Z. Wen, and M. Li. Using support vector machine combined with auto covariance to predict protein-protein interactions from protein sequences. *Nucleic Acids Res.*, 36(9):3025–30, 2008.

[6] H. Lin, X. Ma, P. Chandramohan, A. Geist, and N. Samatova. Efficient data access for parallel BLAST. In *IPDPS*, 2005.

[7] S. Martin, D. Roe, and J. L. Faulon. Predicting protein-protein interactions using signature products. *Bioinformatics*, 21(2):218–226, January 2005.

[8] Y. Park. Critical assessment of sequence-based protein-protein interaction prediction methods that do not require homologous protein sequences. *BMC Bioinformatics*, 10:419, 2009.

[9] S. Pitre, M. Alamgir, J. R. Green, M. Dumontier, F. Dehne, and A. Golshani. Computational methods for predicting protein-protein interactions. *Seitz, H (ed), Advances in Biochemical Engineering/Biotechnology (Springer-Verlag)*, 2008.

[10] S. Pitre, F. Dehne, A. Chan, J. Cheetham, A. Duong, A. Emili, M. Gebbia, J. Greenblatt, M. Jessulat, N. Krogan, X. Luo, and A. Golshani. PIPE: a protein-protein interaction prediction engine based on the re-occurring short polypeptide sequences between known interacting protein pairs. *BMC Bioinformatics*, 7:365, 2006.

[11] S. Pitre, M. Hooshyar, A. Schoenrock, J. Green, F. Dehne, and A. Golshani. Short co-occurring polypeptide regions can predict global protein interaction maps. *submitted*, 2011.

[12] S. Pitre, C. North, M. Alamgir, M. Jessulat, A. Chan, X. Luo, J. R. Green, M. Dumontier, F. Dehne, and A. Golshani. Global investigation of protein-protein interactions in yeast saccharomyces cerevisiae using re-occurring short polypeptide sequences. *Nucl. Acids Res.*, page gkn390, 2008.

[13] J. Shen, J. Zhang, X. Luo, W. Zhu, and K. Yu. Predicting protein-protein interactions based only on sequences information. *Proc. Natl. Acad. Sci. USA*, 104:4337–41, 2007.

[14] C. Yu, L. Chou, and D. Chang. Predicting protein-protein interactions in unbalanced data using the primary structure of proteins. *BMC Bioinformatics*, 11:167, 2010.