International Journal of Computational Intelligence Research
ISSN 0973-1873 Volume 12, Number 2 (2016), pp. 95-102

http://www.ripublication.com

Protein-Protein Interaction Network Using Decoupled Software Pipelining

Narendra Chaudhari and Anjali Mahajan

Asst. Prof. Dept. of I.T.,
Dr. Babasaheb Ambedkar College of Engineering and Research,
Wanadongri, Nagpur-441110, Maharashtra India
E-mail: narendra.chaudhari268@gmail.com
Prof. and Head of Dept. Dept of Information Tech.
Govt. Polytechnic, Sadar Nagpur-440001, Maharashtra India
E-mail: armahajan@rediffmail.com

Abstract

Protein–protein interactions are fundamental functions in living cells. In recent years the data sets of interaction graphs represented in the form of networks has increased. Markov clustering (MCL) is becoming a key algorithm within bioinformatics for determining clusters in networks. Parallelization on multicore architecture and bioinformatics is a great challenge for researchers.

This paper presents a novel parallel implementation of Protein-Protein Interaction(PPI) Network with MCL using Decoupled Software Pipeline algorithm. The computationally intensive algorithm implementation achieves almost linear speedup on sample data set. The experiments on three real PPI network shows significant improvements in quality, balance and execution speed.

Keywords: Multicore, MCL, DSWP, PPI Network, non-speculative Parallelization

1. INTRODUCTION

High throughput measuring devices generate lot of data set for bioinformatics research. The problems involving these data require extensive computational power. One of the key solutions to time-efficient data processing is the proper implementation of computational intensive tasks using parallel technology. The processor performance can be maximized and power efficiency can be increased with

the help of parallelism [1]. A large number of cores is combined into a single chip to improve the overall performance of the multi-core processors. The conventional serial programs are being transformed to exploit the advantage of modern multicore systems. This depicts the current trends in processor architecture.

Many biological functions in living cells require interactions among proteins. They never act as single isolated species to perform their function. Thus Protein – protein interactions are fundamental to every cellular and biological process[2]. Due to this the availability of huge data sets of interaction graphs has increased. This interaction can be naturally represented in the form of networks. These networks give us the initial idea of protein interaction on genomic scale. Only a small fraction of the real interaction networks is actually detected experimentally. Predication of protein interaction through computing have become essential to augment the experimental methods[3]. This is particularly true for sequence based prediction methods which do not require additional data. The data such as homologous sequences or 3D structure information are often not available. Due to this the availability of huge data sets of interaction graphs has increased. This interaction can be naturally represented in the form of networks. These networks give us the initial idea of protein interaction on genomic scale. The development of methods to extract valuable biological knowledge from these networks and understand the basic component of cell machinery has become the most important requirement in this field.

In this paper we report on a novel parallel implementation of Protein-Protein Interaction Network with Markov Clustering (PPIMC) using Decoupled Software Pipeline algorithm (DSWP)[4]. The computationally intensive algorithm implementation achieves almost linear speedup when run on large protein-protein interaction networks. The functionally related protein modules are accurately identified by this algorithm

2. PROTEIN-PROTEIN INETRACTION NETWORK WITH MARKOV CLUSTERING (PPIMC)

Clustering is an efficient approach to understand the relation between the protein-protein interaction network and its function. Clustering in protein interaction networks is to group the proteins into sets (clusters) which demonstrate greater similarity among proteins in the same cluster than in different clusters.

The graph networks used in bioinformatics have cluster information, which is a very important parameter. The Markov clustering, which originally was developed for general graph clustering, is an important bioinformatics algorithm. It is being used in a number of wide ranges of bioinformatics applications. The most important among them is the protein-protein interaction networks [5]. The Markov clustering(MCL) is very fast and effective. It is more tolerant and tough to noise. Due to these properties, it is an attractive algorithm in the formation of clusters from the interaction networks.[6]

The graph used by MCL has stochastic (Markov) matrix associated with it. It is defined by normalizing all columns of the adjacency The algorithm uses two simple algebraic operations, *expansion* and *inflation* which operates on the stochastic

(Markov) matrix. The Markov matrix for a graph G is the normalization of all columns of the adjacency matrix G. The clustering of matrix is the simulation of the flow within the graph. This is done using expansion operations. Expansion does matrix squaring by taking the power of a stochastic matrix using the normal matrix product.

The flow is strong at certain points and week at certain. The clustering process then strengthens the flow to make it more strong where it is strong and weakens it where it is week. This operation is called as inflation operation which corresponds with taking the Hadamard power of a matrix. It is then followed by a scaling step where the resulting matrix corresponds to probability values. The two operations expansion and inflation are executed one after another making the graph more apparent. It then converges to a result with strong flow, also called as clusters.

The MCL expansion operator takes r^{th} power of the matrix *M*, where M is *a* column stochastic matrix, a non-negative matrix with the property that each of its columns sums to 1, as

$$Exp\,(M) = M^r \tag{1}$$

By default, r = 2. The stochastic graph associated with *M* has each column *j* of a stochastic matrix *M* corresponds with node *j*. The matrix entry M_{ij} corresponds with the probability of going from node *j* to node *i*. It is observed that for values of $r > 1$, inflation changes the probabilities associated with the collection of random walks.

For the MCL inflation operation-given a matrix $M \in M \in \mathbb{R}^{m \times n}$, $M \geq 0$ and a number $r \in R$, $r > 0$, the inflation operator $\Gamma_r: \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times n}$ to *M* with power coefficient r is defined by

$$(\Gamma_r\, M)_{ij} = \frac{\left(M_{ij}\right)^r}{\sum_{k=1}^{m}\left(M_{kj}\right)^r}\ ;\ i = 1,..m, j = 1..n \tag{2}$$

$\Gamma_r\, M$ is called the inflation matrix of *M* with a power coefficient r. This inflation process automatically normalizes and creates a new Markov matrix result.. The inflation operator can be altered using the parameter *r*. Increasing this parameter has the effect of making the inflation operator stronger, which in turn increases the granularity or tightness of clusters.

Expansion is carried out to computing random walks of higher length, with many steps. It associates new probabilities with all pairs of nodes. One node is treated as the point of departure and the other is the destination. As it is common to have higher length paths within clusters than between different clusters and there are many ways of going from one node to the other, the probabilities associated with node pairs lying in the same cluster will be relatively large. Inflation will then have the effect of boosting the probabilities of intra-cluster walks and will demote inter-cluster walks. The iteration of expansion and inflation processes result in the separation of the graph into different segments as is indicated by an idempotent matrix. This matrix has clusters in blocks inside[3]. The process goes on till the final idempotent matrix is generated which has almost same values of the matrix elements on the current expansion and inflation iteration compared and the previous iteration. There are no

longer any paths between these segments and the collection of resulting segments is simply interpreted as a clustering. The idempotent condition is numerically achieved when the convergence, the k^{th} column of matrix M in the current iteration is less than a minimum threshold value e (by default $e = 10^{-3}$), for all k.

In a biological sense, it is found that members of a protein groups are more similar to each other than to proteins in another groups. The flow within protein groups is strong due to this property of biological graphs, which is indicated by random walk. It starts at any given point in a groups and is seen lingering within this groups than to cross to another groups. With this, the MCL algorithm[3] can be written as shown in Figure 1:

The largest computation time involved is in the expansion, inflation and convergence of the algorithm. This increases further with the size of the Markov matrix. In PPI, the networks are generally sparse, so sparse matrix can be used for storing the values. We can improve the performance of the algorithm by implementing parallel tasks for the two algebraic operations-expansion and inflation and computation of the convergence function. Most extensive computation is required for the expansion operation.

The expansion operation is basically matrix multiplication in which many parallel algorithms exist. The programs for expansion and inflation are shown in Figure 2 and Figure 3.

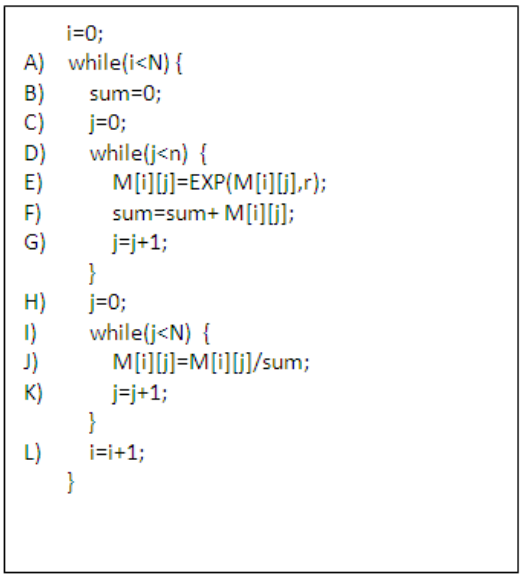
We apply Decoupled Software Pipelining (DSWP) technique to expansion operation. This is very effective in computations which involve recursive data structures, which in fact this operation has. In this technique a single-threaded sequential loop is split into multiple threads, each of which performs useful computation necessary for overall program correctness.

```
G is a graph
add loops to G
set Γ to some value          # affects granularity
set M_1 to be the matrix of random walks on G
while (convergence < threshold) {
    M_2 = M_1 * M_1   # expansion
    M_1 = Γ(M_2)      # inflation
    convergence = difference(M_1, M_2)
  }
```

Figure 1: MCL algorithm

```
      i=0;
A)    while(i<N) {
B)          j=0;
            while(j<N) {
C)            sum=0;
D)            k=0;
E)            while(k<N) {
F)              sum=sum+A[i][k]*M[k][j];
G)              k=k+1;
              }
H)            M_Temp[i][j]=sum;
I)            j=j+1;
            }
J)          i=i+1;
      }
K)    i=0;
L)    while(i<N) {
M)          j=0;
N)          while(j<N) {
O)            M[i][j]=M_Temp[i][j];
P)            j=j+1;
            }
Q)            i=i+1;
      }
```

Figure 2: Program for Expand



```
     i=0;
A)   while(i<N) {
B)     sum=0;
C)     j=0;
D)     while(j<n) {
E)       M[i][j]=EXP(M[i][j],r);
F)       sum=sum+ M[i][j];
G)       j=j+1;
       }
H)     j=0;
I)     while(j<N) {
J)       M[i][j]=M[i][j]/sum;
K)       j=j+1;
       }
L)     i=i+1;
     }
```

Figure 3: Program

3. DECOUPLED SOFTWARE PIPELINING

The DSWP is an effective technique for parallelization of applications. It partitions the loop body into stages of a pipeline where the critical path dependencies are local to a stage with each stage executing within a thread. This avoids communication latency on the critical path. There are four steps of DSWP operation[8]. They are-

- Build the Program Dependency Graph(PDG) of the loop. The DAG contains all data (both register and memory) and control dependences, both intra-and inter iteration. The solid line shows data dependency and dashed lines control dependency. Figure 4 and Figure 5 shows DAG for the expansion and inflation operations.
- Group dependence cycles into strongly-connected components (SCCs) that form an acyclic graph. Figure 6 and Figure 7 shows the DAG_{SCC} for the operations. Each node in the DAG_{SCC} is a single SCC. The cross-thread cyclic dependencies are eliminated[9]. These SCCs form the minimum scheduling units.
- Allocates each SCC to a thread. There may be more SCC than the number of threads available. The SCC's are merged until they match the number of threads. DSWP extracts at most one thread per SCC. DSWP tries to balance the load on each stage when assigning SCCs to threads[10].
- Insert synchronization. To transmit data values in case of data dependences and branch conditions for control dependences synchronization is required. The core-to-core communication overhead must be as low as possible.

The success of DSWP depends on being able to extract the relatively fine grain

parallelism that is present in many applications.

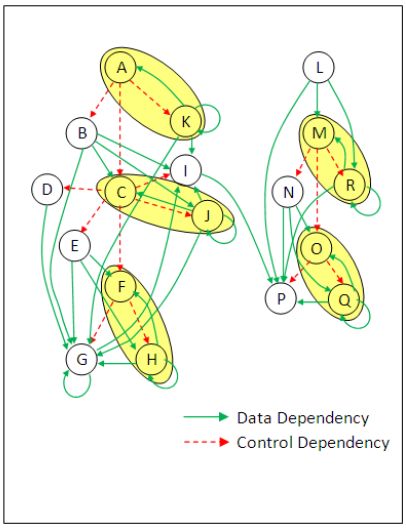


Figure 4: PDG for program in Fig. 2

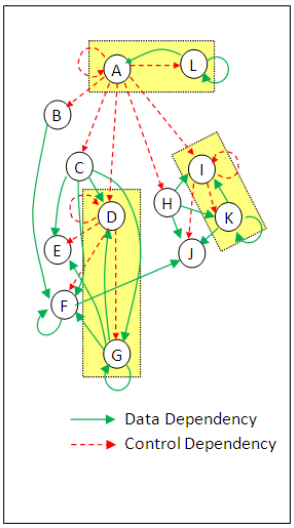


Figure 5: PDG for program in Fig 3

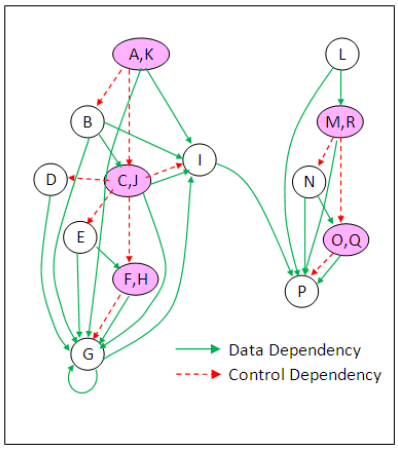


Figure 6: SCC for Figure 4

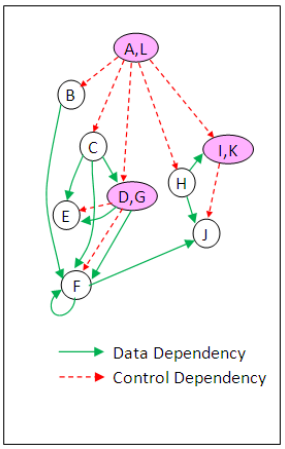


Figure 7: SCC for Figure 4

4. RESULTS

The proposed algorithm was evaluated using three real protein-protein interaction networks as shown in Table. The PPI network of Yeast is obtained from the Database of Interacting protein (DIP), another PPI network of Yeast is obtained from other database called BioGRID database. The third network is a Human PPI network, obtained from the iRefIndex.

The largest computation is involved in the expansion and inflation routines. It requires large storage to store the large network data when using the expansion routine. In the proposed algorithm for PPIMC, it is proposed to implement parallel tasks for both expansion and inflation. In PPI sparse matrices are used to store the protein-protein

interaction.
The table 1 below shows the PPI data set used in the experimental part for testing the proposed algorithm.

Table 1: List of Protein Samples

Organism	Source	Proteins	Interactions
Yeast-PPI1	DIP	3682	12267
Yeast-PPI2	BioGRID	7846	87981
Human-PPI3	iRefIndex	12870	54628

The DSWP transformation is applied to PPIMC for parallelizing the algorithm. In DSWP the delay due to communication latency is totally avoided as the loop critical path dependence is not required to be routed from core to core to achieve pipelined parallelism. DSWP breaks the loop iteration instead of placing each iteration alternately on each core. This allows the algorithm to properly utilize the cores in multicore systems.
The performance of sequential and the proposed algorithm where DSWP is applied to PPIMC is presented below. The proposed algorithm performs better than the sequential version. PPI1 has shown speedup of 4.79 %, PPI2 by 6.65 and PPI3 has 9.22. The proposed algorithm with DSWP provides noticeable performance improvement. It is limited by the number and size of the strongly connected components. DSWP extracts at most one thread per strongly connected components. The improvement is also because DSWP sometimes uses instruction level parallelism for thread level parallelism.
The speedup achieved on the three operations is shown in figure 9.
The proposed algorithm shows speedup upto 8 times for expansion, 5 times for Inflation and 4 times for convergence. This algorithm has been proven to be essential to the adaptation of the grain of parallelism to the target and to the effectiveness of load balancing on various threads.

Figure 8: Result of various PPI

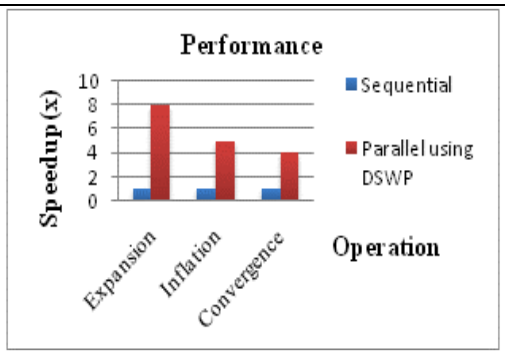


Figure 9: Performance of operation in MCL PPI

5. CONCLUSION

This paper introduces a parallel algorithm for Protein-Protein Interaction using Markov Clustering transformed with Decoupled Software Pipelining. The proposed algorithm achieves significant speedups over the current best sequential programs and has accuracy comparable to them. Results also illustrate that in DSWP the delay due to communication latency is totally avoided as the loop critical path dependence is not required to be routed from core to core to achieve pipelined parallelism. DSWP breaks the loop iteration instead of placing each iteration alternately on each core. This allows the algorithm to properly utilize the cores in multicore systems.

6. REFERENCES

[1] T.P. Chen and Y.-K. Chen, "Challenges and Opportunities of Obtaining Performance from Multi-Core CPUs and Many-Core GPUs," IEEE Int'l Conf. Acoustics, Speech, and Signal Processing, pp. 613-616, 2009.

[2] F. Va´zquez, G. Ortega, J. Ferna´ndez, and E. Garzo´n, "Improving the Performance of the Sparse Matrix Vector Product with GPUs,"Proc. 10th IEEE Int'l Conf. Computer and Information Technology(CIT '10), 2010.

[3] S. Brohe´e and J. van Helden, "Evaluation of Clustering Algorithms for Protein-Protein Interaction Networks," BMC Bioinformatics, vol. 7, article 488, 2006.

[4] G. Ottoni, R. Rangan, A. Stoler, and D. I. August. Automatic thread extraction with decoupled software pipelining. Microarchitecture, IEEE/ACM International Symposium on, 0:105–118, 2005.

[5] B. Schwikowski, P. Uetz, and S. Fields, "A Network of Protein-Protein Interactions in Yeast," Nature Biotechnology, vol. 18, pp. 1257-1281, 2000.

[6] A. Thomas, R. Cannings, N. Monk, and C. Cannings, "On the Structure of Protein-Protein Interaction Networks," Biochemical Soc. Trans., vol. 31, pp. 1491-1496, 2003.

[7] S. Brohee, Van Helden J., Evaluation of clustering algorithms for protein-protein interaction networks [J], BMC Bioinformatics, 7(1), 2006, 488.

[8] H. Zhong, M. Mehrara, S. Lieberman, and S. Mahlke. Uncovering hidden loop level parallelism in sequential applications. In *Proc. Of the 14th International Symposium on High-Performance Computer Architecture*, 2008.

[9] C. Zilles and G. Sohi. Master/slave speculative parallelization. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 85–96, November 2002.

[10] J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst., 9:319–349, July 1987.

[11] E. Raman, G. Ottoni, A. Raman, M. Bridges, and D. I. August. Parallel-stage decoupled software pipelining. In *Proceedings of the2008 International Symposium on Code Generation and Optimization*, April 2008.