Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Performance Enhancement of Smith-Waterman Algorithm Using Hybrid Model: Comparing the MPI and Hybrid Programming Paradigm on SMP Clusters

Mahdi Noorian
School of Computer Science
University of New Brunswick
Fredericton, Canada

Hamidreza Pooshfam
School of Computer Science
University Sains Malaysia
Pulau Pinang, Malaysia

Zeinab Noorian
School of Computer Science
University of New Brunswick
Fredericton, Canada

Rosni Abdullah
School of Computer Science
University Sains Malaysia
Pulau Pinang, Malaysia

***Abstract*— Nowadays, database pattern searching is the most heavily used operation in computational biology. Indeed, sequence alignment algorithm plays an important role to find the homologous groups of sequences which may help to determine the function of new sequences. Meanwhile Smith-Waterman algorithm is one of the most prominent pattern matching algorithms. However, it cost the large quantity of time and resource power. By the aid of parallel hardware and software architecture it becomes more feasible to get the fast and accurate result in efficient time. In this paper, Smith-Waterman algorithm is parallelized base on various types of parallel programming, pure MPI, pure OpenMP and Hybrid MPI-OpenMP model. In addition, based on the experiments it will be proved that hybrid programming which employ the coarse grain and fine grain parallelization, is more efficient compare with pure MPI and pure OpenMP in cluster of SMP machines.**

***Keywords*—Pattern Matching, Smith-Waterman Algorithm, Hybrid MPI-OpenMP.**

I. INTRODUCTION

With the emergence of parallel hardware and software technologies, developers are encountered with the challenge of choosing a programming paradigm best suited for the underlying computer architecture [12]. Generally, Parallel computers can be divided into two main categories: Shared Memory Processor (SMP) and Distributed-Shared Memory Processor (DSMP) [13]. Due to the fact that using more powerful machines with SMP architecture has high hardware cost, thus the manufacturer intend to combine low-cost SMP machines together as a cluster of SMPs (CLUMPs) [13,1] which is the domain of focus in this paper.

On the other hand, by generating high amounts of data in new biological experimental techniques generic sequence searching becomes one of the most heavily used operations in computational biology [11,15,9]. In particular, the size of GenBank/ EMBL/ DDBJ double every 15 months [3]. Therefore analyzing generic databases with such a constant growth, raise a challenge for scientist, in respect of being time consuming, expensive and impractical [14]. Smith-Waterman algorithm [18] is one the most significant and widely-used similarity algorithms for biological sequence comparison that adopts the dynamic programming method [11].

Despite its high sensitivity in identifying best local alignments, it is very time consuming and computationally expensive process. This algorithm requires quadratic time for each comparison of two sequences [16]. Therefore, because of the complexity of this algorithm, there is a need for a methodology that could reduce the computation time while delivering accurate results.

In this study, various programming paradigms are being compared for the parallelization of Smith-Waterman algorithm on a cluster of SMP nodes.

Specifically, an evaluation between different type of implementation paradigm such as pure MPI, pure OpenMP and Hybrid model which is a combination of both, is provided in terms of execution time.

The remainder of this paper is organized as follows. In the next section the different parallelization approach is presented. In section III the original algorithm will be described in details while in section IV the methodology for hybrid model and coarse grain and fine grain parallelization will be discussed. Section V explains the experimental results and comparisons and finally the last section will conclude the paper.

II. PARALLELIZATION APPROACH

Nowadays, there are numerous approaches which are suitable for transforming sequential Smith-Waterman algorithm into parallel paradigms. This transformation can be performed by using MPI, OpenMP or hybrid model. Based on the hardware architecture either of aforementioned models needs its own library and runtime support systems [20]. In following subsections each methodology will be described.

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

A. MPI and OpenMP

Message Passing Interface (MPI) is a standard for inter-process communication for distributed-memory multiprocessor application. When an MPI program starts, the program spawns into the number of processes as specified by the user. Each process runs and communicates with other instances of the program, possibly running on the same processor or different one.

On the other side, OpenMP is an open standard for providing parallelization mechanisms on shared-memory multiprocessors. The standard provides a specification of compiler directives, library routines, and environment variables that control the parallelization and runtime characteristics of a program

B. Hybrid; MPI + OpenMP

As can be seen in Figure 1, the hybrid programming model provides opportunity to take advantage of the both abovementioned models at the same time. The hybrid programming model instinctively matches with the structural characteristics of a cluster of SMP nodes, as well as providing two level communication patterns: intra and inter-node communication [7].

To elaborate, Intra-node communication is feasible by the aid of OpenMP and thread programming model. On the other hand, Intra-node communication happens when multiple threads has common access to the one particular node's share memory. The main concern in this common access is to provide synchronization to ensure data consistency throughout the execution of the program which is supported by OpenMP.

Subsequently, inter-node communication is achieved through message passing Interface technology between nodes.

III . SMITH-WATERMAN ALGORITHM

The Smith-Waterman algorithm which was enhanced by Gotoh [9] is perhaps the most widely-used local similarity algorithm for biological sequence pair wise alignment. In Smith-Waterman database search, the dynamic programming method is used to compare every database sequence to query sequence and assign a score to each comparison result [14]. This algorithm is built on the idea of comparing segments of all possible lengths between two sequences to find the best local alignment. This means that the algorithm is very sensitive and accurate which guarantee to locate an optimal alignment of the sequences [4].
The algorithm consists of three steps:

1. Fill the dynamic programming matrix.
2. Find the maximal value in the matrix.
3. Trace back the path that leads to maximal score to find the optimal local alignment.

To elaborate, let's assume that the first/main sequence, *S*, contains characters of length *m*, and the pattern sequence, *T*, is the length of *n*.

$$S = \{s_0, s_1, s_2 \ldots s_{m-1}\}$$
$$T = \{t_0, t_1, t_2 \ldots t_{n-1}\}$$

In the first step, a two-dimensional similarity matrix *R* is created by considering *m+1* as the number of rows and *n+1* as the number of columns. The first row and the first column are assigned the value zero. Hence, filling up the cells will start from the left topmost empty cell which is $R_{(1,\ 1)}$, continued by the cell on its right, $R_{(1,\ 2)}$ and it will proceed to cell $R_{(m+1)\ (n+1)}$.

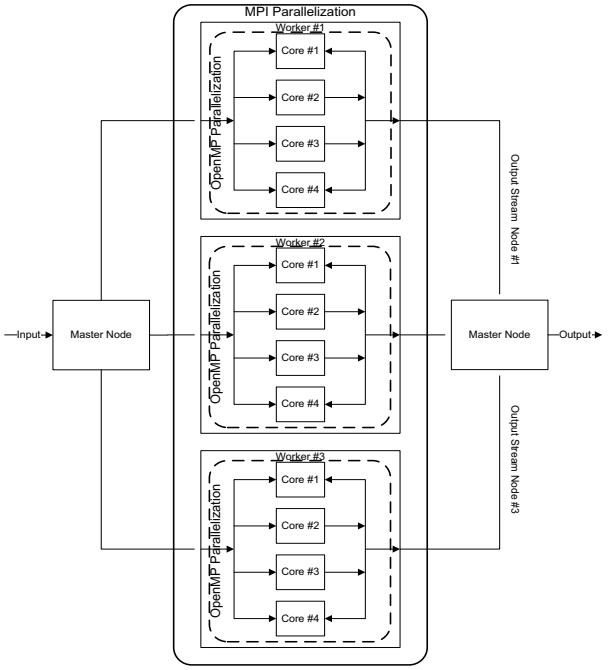

Figure.1 Hybrid model, MPI and OpenMP level of parallelization

Subsequently, in the second step, the score of each cell is determined by choosing the maximum score resulted from equation (1). Assume that, the gap penalty is presented by *g* and similarity score with *sbt*.
The equations are:

$$R_{i\,j} = \max \begin{cases} R_{(i-1)(j-1)} + Sbt_{(s_i,t_j)} \\ R_{(i-1)j} + g \\ R_{i(j-1)} + g \\ 0. \end{cases} \quad (1)$$

Where the conditions are:

- *if* $(s_i = t_j)$ *then* $Sbt(s_i, t_j) = 2$
  else -1
- g = -2

Finally, after all cells are filled up, a search for highest value in the matrix will be performed. Then, by referring to the location of the highest value, it starts to trace back the operation by taking into account the vicinity of the value which is either the upper, left, or right cell. Applying the above equation indicates that, the complexity of this search algorithm is *O(mn)*. As it is mentioned before, the drastic growth in bioinformatics data cause an increase to *O(kmn)* steps where *k* represents the exponential growth of the size in genetic databases [11].The optimized approach to overcome this challenge is to implement this algorithm via parallel programming solution by taking advantage of both MPI and OpenMP technologies.

IV. The Approach & the Methodology

In order to implement this algorithm either the task or the data can be divided between processors. Task decomposition is about to break down the job in different parts and assign each part of job to a specific processor while in data decomposition, all processors apply the same job on different portion of data. As matter of fact, in most cases the nature of algorithm and study would determine which type of decomposition has to take place. For this algorithm, data decomposition parallel approach is utilized in both node level and cluster level.

A. Coarse Grain Parallelism Using MPI at Cluster Level

In practice, a target database is partitioned and distributed between different cluster nodes by the aid of MPI technology. Afterward a similarity matching algorithm is applied on the data to search for the best alignment. Since in this study the FASTA file is being used as input data, there is a requirement for some pre-processing and data manipulation phase before the algorithm could be applied on them. Therefore the Master node is responsible to perform this prerequisite action in order to omit irrelevant data and extract the pure sequences and make them ready for further process. The detailed description of the Master node's job is depicted in Table I.

TABLE I
MASTER NODE RESPONSIBILITIES

1. Read the data from FASTA file.
2. Separate each sequence and remove the description info form them; add the unique ID to each sequence.
3. Assign each sequence to each slave base on *load balancing technique.*
   for (i = 1 to number of slaves)
   a) find the smallest bucket (in matter of number of characters)
   b) Append pure sequence to bucket(i)
4. for (i =1 to number of slave)
   Send (bucket[i] to slave[i]);
5. Get query sequence and broadcast to slaves.
6. Receive (result from slaves).
7. Combine the result and Output.

B. Applying Dynamic Load Balancing

One of important factor that may degrade a performance is overhead that happens as a result of unload balanced data distribution among processors. Uneven load distribution may cause some processors terminate their job earlier than the other; as a result, the processes with a lighter load will remain idle while those with a heavier load are trying to finish their tasks. Indeed this matter can adversely affect on the total execution time and consequently the speed up.

As aforementioned, master node has the responsibility of data decomposition and divide the database sequences between slave nodes. In the other hand, database sequences have the different length which may vary from 500 up to 2000 characters. Therefore, it seems necessary to have the method which handles the sequences distribution not only based on the number of sequences. Thus, the dynamic method is proposed which can help to decompose the data in more balance manner. Unlike static load balancing technique which only distributes the target database based on the number of slaves, this method divide sequences among the slaves based on the number of characters that each sequence contains. To illustrate, in this proposed method it is assumed that each slave nodes have their own *Bucket[i]*, *i* refer to the slave node's number, and the database have *k* sequences, which is $0<k\leq n$, the load balancing is performed as following steps :

1. Assign the first sequence to *Bucket[1]*, and second sequence to *Bucket[2]*, and sequence *i* to *Bucket[i]*.
2. Get each *Bucket[i]* size and assign the new sequences to the one who has the smallest size.
3. Continue the step 2 until all database sequences divide between the Buckets.

In fact, this model can ensure that each slave will receive the same amount of data thus it could help to reduce the idle time of each slave processor and increase the performance.

C. Fine Grain Parallelism Using OpenMP at Node Level

As can be seen in Figure 2, besides using MPI at the cluster level, another level of parallelization will be performed inside the slave nodes by the aid of OpenMP technology. To clarify, several threads are spawned inside slave nodes and each of them applies the Smith-Waterman algorithm on different segments of data. The detailed responsibility of slave nodes will illustrated in Table II.

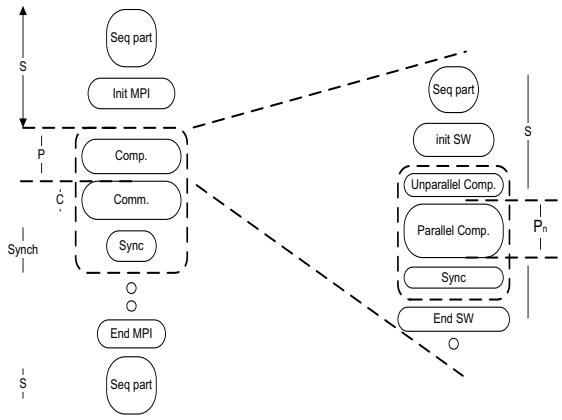

Figure.2 Parallelizing a MPI code with OpenMP by using fine grain approach.

TABLE II
SLAVE NODE RESPONSIBILITIES

1. Receive (bucket[i] form Master)
2. Receive (query sequence)
3. for (i=1 to bucket size)
   Smith-Waterman(bucket[i], query sequence);
4. Send (result to Master).

V. EXPERIMENT AND ANALYSIS

Based on abovementioned methodology, Smith-Waterman algorithm has been implemented using MPI and OpenMP and Hybrid paradigms. The performance of this parallel implementation was evaluated using various database sizes and a against specific query sequences.

A. Experiment Infrastructure

The hardware used for the experiments used Dell PowerEdge cluster which consists one 2x Quad-Core Intel Xeon 1.6GHz, 2x4MB Cache as master and two 2x Quad-Core Intel Xeon 1.6GHz, 2x4MB Cache as worker running on PureOS 64bit. Both clusters are interconnected using the fast Ethernet by Gigabyte switch.

B. Experiment Result

This study conducts different classes of experiments for the homology searches in three parallel implementation model based on the number of processors and database size.

The data provided by ExPASy Proteomics [19]. The database size starts from 83KB (approximately 200 sequences) up to 175MB (about 102400 sequences).

As a first class of experiments, the execution time for the three programming paradigm is calculated (sequential, parallel MPI and Hybrid model) across different size of data.

As it is depicted in Figure 3, the hybrid model which get benefit from both MPI and OpenMP technology, obtain better result and performance in terms of execution time compare with pure MPI parallel implementation and clearly sequential model. Note that, the sequential implementation of the algorithm cannot be executed for 102400 numbers of sequences. In this experiment, six quad-core processors are being used as our computational resources.

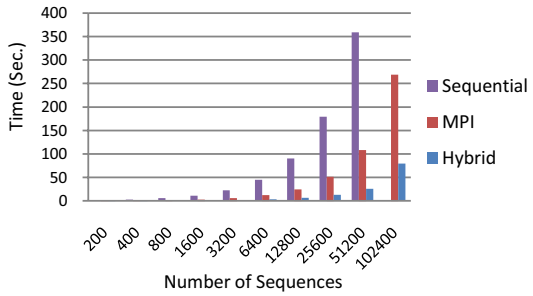

Figure.3 Comparison between three methods for different data files with 6 CPUs

Subsequently for the second class of experiments, it is focused on *speed up* feature of parallel processing.

To illustrate, Figure 4 displays the speed up diagram which prove the better performance of Hybrid Model against Pure MPI. Based on experimental results the output for hybrid model significantly has better speed up in compare with pure MPI model. What is clear from the curve by implementing hybrid model in compare with pure MPI implementation is that in CLUMPS machine with multi core processor hybrid implementation can show almost fourteen-fold speed up with six quad core processors in master-slave model. On the other hand, due to the fact that, in hybrid model the whole process is divided in two levels: the node levels and the CPU level, in both levels the process take place concurrently and it helps to get more efficient speed up in whole process.

With regard to results that gathered from experiment and its visual presentation in Figure 4 and Figure 5 maximum speed up which achieved with 6 CPUs is equal to 13.962.

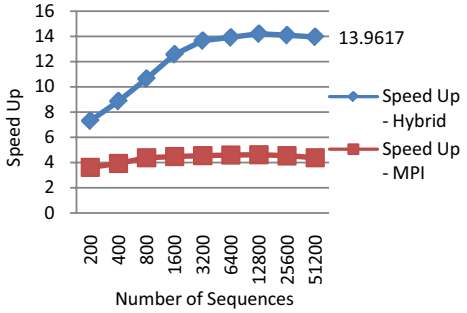

Figure. 4 Comparison in speed up with six CPUs

In addition, Figure 5 demonstrates speed up of MPI and Hybrid model for a specific input data across different processor number.

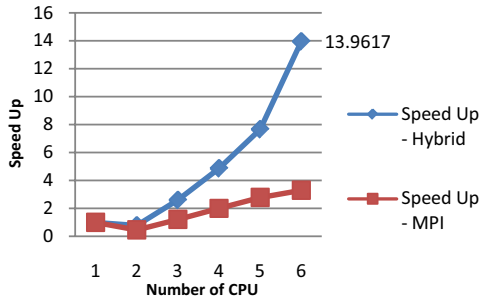

Figure. 5 Comparison speed up between MPI and Hybrid methods for data file with 3200 Sequences

So far, the advantages of hybrid model against pure MPI are examined. In the last class of experiments it is required to look into the functionally and performance of pure OpenMP in details.

Figure 6 exhibits the performance of pure OpenMP with different inputs database size. To elaborate, Pure OpenMP get advantages from Share Memory Structure along with low-light thread programming. Evidently, the computational power inside the nodes increases considerably due to the thread programming. On the other hand, share memory structure along with OpenMP enables it to get benefit of Inter-Node communication rather than Intra-Node communication that exist in pure MPI. This makes the communication overhead in OpenMP rather negligible compare with the one in pure MPI.

It is noteworthy to mention that, the cost of OpenMP is mainly calculated based on thread initialization and synchronization while in MPI the main cost is imposed by communication overhead.

To conclude, pure OpenMP achieve almost seven-fold speed up with two quad core processors while in pure MPI this performance is only four fold.

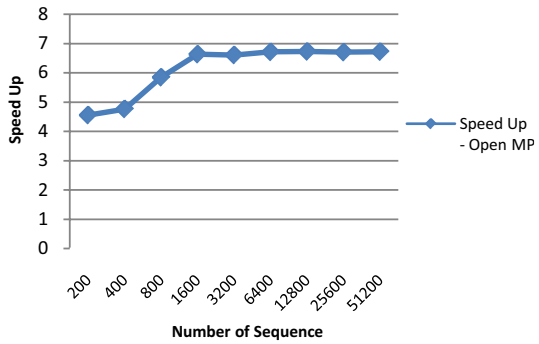

Figure. 6 Pure OpenMP speed up with two CPUs

VI. Conclusion

In this paper, the hybrid implementation of the Smith-Waterman algorithm is presented, which combines fine grain and coarse grain parallelism and multi-level scheduling. This implementation achieved a speed up fourteen on a cluster five Quad-Core Intel Xeon 1.6GHz as workers and one as master. Nowadays, it becomes an obligation to use hybrid implementation by taking advantages from multi-core processors technology in clusters of SMP machine. Since a processor in a cluster contains of different cores and each core can run a thread separately, these types of clusters can give a significant speed up with hybrid implementation as compared with pure MPI. Breaking job in to the threads at the processor level and using share memory with high speed bus connections gives us the opportunity to decrease the execution time in hybrid model.

References

[1] Abdul Rashid, N., Abdullah, R., & Zawawi, A. H. (2007). Parallel homologous search with Hirschberg algorithm: a hybrid MPI-Pthreads solution. *Proceedings of the 11th WSEAS International Conference on Computers* (pp. 228-233). Agios Nikolaos, Crete Island, Greece: World Scientific and Engineering Academy and Society (WSEAS).

[2] Adhianto, L., & Chapman, B. (2006). Performance Modeling of Communication and Computation in Hybrid MPI and OpenMP Applications. *Proceedings of the 12th International Conference on Parallel and Distributed Systems - Volume 2* (pp. 3-8). EEE Computer Society.

[3] Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., & Wheeler, D. L. (2008). GenBank. *Nucleic Acids Research* , 25-30.

[4] bio, C. (2007, May 1). Retrieved July 14, 2008, from CLC bio: www.clcbio.com

[5] Boukerche, A., Melo, A. C., Sandes, E. F., & Ayala-Rincon, M. (2007). An exact parallel algorithm to compare very long biological sequences in clusters of workstations. *Cluster Computing* (pp. 187-202). Hingham, MA, USA: Kluwer Academic Publishers.

[6] Capello, F., & Etiemble, D. (2000). MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. *Conference on High Performance Networking and Computing* (p. 12). Dallas, Texas, United States: IEEE Computer Society.

[7] Drosinos, N., & Koziris, N. (2004). erformance comparison of pure MPI vs hybrid MPI-OpenMP parallelization models on SMP clusters. *18th Int. Parallel & Distributed Symposium*, (p. 15).

[8] Farrar, M. (2007). Striped Smith--Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics* (pp. 156-161). Oxford University Press.

[9] Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of Molecular Biology , 162*, 705-708.

[10] Guan, X., Mural, R. J., & Uberbacher, E. C. (1995). Sequence comparison on a cluster of workstations using the PVM system. *9th International Parallel Processing Symposium* (pp. 190-195). ipps.

[11] Hsien-Yu, L., Meng-Lai, Y., & Yi, C. (2004). A parallel implementation of the Smith-Waterman algorithm for massive sequences searching. *Engineering in Medicine and Biology Society, 2004. IEMBS apos;04. 26th Annual International Conference of the IEEE*, (pp. 2817-2820). San Francisco, CA, USA.

[12] Jost, G., Jin, H., Mey, D. a., & Hatay, F. F. (2003). *Comparing the OpenMP, MPI, and Hybrid Programming Paradigms on an SMP Cluster*. Germany: NAS Technical Report.

[13] Matin, C. (2004, April). Programming Options for Distributed Shared Memory Cluster Computers. *Center for High Performance Computing* , pp. 1-3.

[14] Meng, X., & Chaudhary, V. (2005). Exploiting Multi-level Parallelism for Homology Search using General Purpose Processors. *Proceedings of the 11th International Conference on Parallel and Distributed Systems - Workshops (ICPADS'05) - Volume 02* (pp. 331-335). Washington, DC, USA: IEEE Computer Society.

[15] Murakami, M. M., Maria, E., Walter, M. T., & Martins, W. S. (2003). Parallel Implementation of the Smith-Waterman Algorithm for Large Scale Database Search. *The 1st International Conference on Bioinformatics and Computational Biology - ICoBiCoBi, 2003.* Ribeirão Preto.

[16] Sanchez, F., Salami, E., Ramirez, A., & Valero, M. (2005). Parallel processing in biological sequence comparison using general purpose processors. *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, (pp. 99-108).

[17] Smith, L. A. (2001). Mixed Mode MPI/OpenMP Programming. *UKHEC* . Edinburgh: Edinburgh Parallel Computing Center.

[18] Smith, T. F., & Waterman, M. S. (1981). Identification of Common Molecular Subsequences. *Journal of Moleculel Biology , 147*, 195-197.

[19] *Swiss-Prot*. (2008). Retrieved 2008, from ExPASy Proteomics Server: http://www.expasy.ch/sprot/

[20] Thaker, D., Sun, L., Jiang, H., Hase, W. L., & V., C. (2002). Experiments with Parallelizing a Tribology Application. *Proceedings of the 2002 International Conference on Parallel Processing Workshops* (p. 344). Washington, DC, USA: IEEE Computer Society.