International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-4 Issue-7, December 2014

Parallel Implementation of Smith-Waterman Algorithm using MPI, OpenMP and Hybrid Model

Zeiad El-Saghir, Hamdy Kelash, Sayed Elnazly, Hossam Faheem

Abstract*—Pairwise sequence alignment is often used to reveal similarities between sequences, locate patterns of conservation, study gene regulation, and infer evolutionary relationships [1]. Although the Smith–Waterman is the only algorithm guaranteed to find the optimal local alignment, it is also the slowest one as it costs O(mn) for computation & space. Also the volume of biological data is doubling about every six months so the total cost is O(kmn) where k is the size of the database [2, 3]. By using parallel hardware and software architecture accurate results can be achieved in reasonable time. In this paper we show a comparative study for parallelizing smith-waterman algorithm using different parallel models, pure MPI, pure OpenMP and hybrid MPI/OpenMP model. Based on the results it will be proved that hybrid programming which employ the coarse grain and fine grain parallelization, is more efficient compared with pure MPI and pure OpenMP.

***Index Terms* - Smith-Waterman algorithm; MPI; OpenMP; Hybrid MPI/OpenMP; bio-informatics; parallel programming.**

I. INTRODUCTION

Database searches using the optimal algorithm are unfortunately quite slow on ordinary computers; so many heuristic alternatives have been developed, such as FASTA and BLAST. These methods have reduced the running time by a factor of up to 40 compared with the best-known Smith-Waterman implementation, however, at the expense of sensitivity. As a result, a distantly related sequence may not be found in a search using these heuristic algorithms. The use of parallel computers has brought some hope on improving the performance of the pairwise sequence comparison operation when using the Smith-Waterman algorithm. Parallel computers can be broadly divided according to the memory architecture as multicomputer systems with distributed memory and multiprocessor systems with shared memory. The coding of a parallel program for a given algorithm is strongly influenced by the parallel computing system to be used [6].In 2009, dual-core and quad-core processors become standard for normal desktop computers, and chip manufacturers have already announced the introduction of oct-core processors for 2010. It can be predicted from Moore's law that the number of cores per processor chip wills double every 18–24 months [6]. These advances make it available to construct clusters using low priced multicore commodity computers, the architecture chosen in this paper. In this paper we will compare the application of various parallel programming models on the smith-waterman algorithm on a cluster of shared memory computers, including MPI, Open Mp and Hybrid MPI/Open MP in terms of the execution time. The remainder of the paper is organized as follows: section II presents the parallel programming models used in this paper. In section III, the smith-waterman algorithm is discussed. In section IV, our parallel approach and methodology is discussed. In section V results are presented and analyzed and the paper is concluded in section VI.

Manuscript published on 30 December 2014.
*****Correspondence Author(s)
Dr. Zeiad El-Saghir, Department of Computer Science and Information, Majmaah University/ College of Science at Zolfi/ Saudi Arabia.
Eng. Sayed Elnazly, Department of Computer Science and Engineering, Menoufia University/ Faculty of Electronic Engineering/ Egypt.
Assistant Prof. Hamdy Kelash, Department of Computer Science and Engineering, Menoufia University/ Faculty of Electronic Engineering/ Egypt.
Prof. Hossam Faheem, Department of Computer Systems, Ain Shams University/ Faculty of Computers and Information. Egypt.

II. PARALLEL PROGRAMMING MODELS

As mentioned in the previous section query sequence is compared separately with each sequence in the database which can be implemented using message passing paradigm and as we will show in the following section the independency in calculating the anti-diagonal cells in the similarity matrix make it appropriate for implementation using threading paradigm . Whilst mixed mode codes may involve other programming languages such as High Performance Fortran (HPF) and POSIX threads, MPI and OpenMP represent industry standards for distributed and shared memory systems respectively [7]. In the following subsections each of these models is discussed.

A. MPI

The Message-Passing Interface (MPI) is a standardization of a message-passing library interface specification. MPI defines the syntax and semantics of library routines for standard communication patterns. Language bindings for C, C++, Fortran-77, and Fortran-95 are supported. Freely available MPI libraries are MPICH, LAM/MPI and OpenMPI [8, 9, 10]. An MPI program consists of a collection of processes that can exchange messages. Normally, each processor of a parallel system executes one MPI process, and the number of MPI processes started should be adapted to the number of processors that are available. Typically, all MPI processes execute the same program in an SPMD style [11].

B. OpenMP

Shared memory opens the possibility to have immediate access to all data from all processors without explicit communication. a joint effort was made by compiler vendors to establish a standard in this field, called OpenMP [12].

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

OpenMP is a set of compiler directives that a non-OpenMP-capable compiler would just regard as comments and ignore. Hence, a well-written parallel OpenMP program is also a valid serial program [13]. The specification consists also of library routines and environment variables which control the runtime characteristics of the program.

C. Hybrid MPI/OpenMP

By utilizing a mixed mode programming model we should be able to take advantage of the benefits of both models. For example a mixed mode program may allow us to make use of the explicit control data placement policies of MPI with the finer grain parallelism of OpenMP [7]. In mixed mode model two level of communication pattern are used, inter-node & intra-node communication. intra-node communication is implemented through common access to each node's shared memory and inter-node communication is achieved through message passing between different nodes [14]. Figure 1 shows a hybrid program consist of two MPI processes communicate through the interconnection network, each fork four OpenMP threads which communicate through the shared memory inside each node.

III. SMITH-WATERMAN ALGORITHM

The Smith-Waterman algorithm is a dynamic programming method for determining similarity between nucleotide or protein sequences. The algorithm was first proposed in 1981 by Smith and Waterman and is identifying homologous regions between sequences by searching for optimal local alignments. To find the optimal local alignment, a scoring system including a set of specified gap penalties is used [Smith and Waterman, 1981]. Homology identified by sequence database searches often implies shared functionality between sequences and further research and development might depend on the accuracy of the search results. The Smith-Waterman algorithm is build on the idea of comparing segments of all possible lengths between two sequences to identify the best local alignment. This means that the Smith-Waterman search is very sensitive and ensures an optimal alignment of the sequences. Unfortunately, this also has the effect that the method is both time and CPU intensive [15]. When obtaining the local alignment, a matrix $H_{i,j}$ is used to keep track of the degree of similarity between the two sequences to be aligned (A_i and B_j). Each element of the matrix $H_{i,j}$ is calculated as in (1):

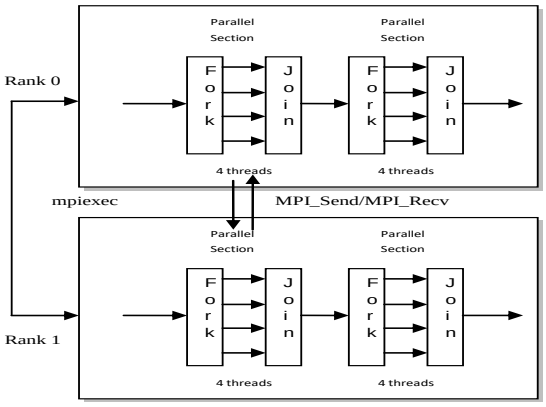

Figure 1. MPI/OpenMP program execution on two nodes.

$$H_{i,j} = \max \begin{cases} 0 \\ H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - d \\ H_{i,j-1} - d \end{cases} \quad (1)$$

Where $S_{i,j}$ is the similarity score of comparing sequence A_i to sequence B_j and d is the penalty for a mismatch. The whole algorithm is divided into three steps:

1. Initialization step
2. Matrix fill step
3. Trace back step

The matrix is first initialized with $H_{0,j} = 0$ and $H_{i,0} = 0$, for all i and j. This is referred to as the initialization step. After the initialization, a matrix fill step is carried out using Equation 1, which fills out all entries in the matrix. The final step is the trace back step, where the scores in the matrix are traced back to inspect for optimal local alignment. The trace back starts at the cell with the highest score in the matrix and continues up to the cell, where the score falls down to a predefined minimum threshold. In order to start the trace back, the algorithm requires finding the cell with the maximum value, which is done by traversing the entire matrix.

As an example, the S-W algorithm, is used to compute the optimal local alignment of two sequences (i.e., A = G A A T T C A G T T A and B = G G A T C G A). Assume that:

$$S_{i,j} = \begin{cases} +5 \text{ if } (A_i = B_i) \\ -3 \text{ else} \end{cases}$$
$$d = -4$$

Table I illustrates the calculation of the DP matrix H and the trace back path (shown in bold red digits). The best score found in the matrix is 14, and the corresponding optimal local alignment is

G A A T T C – A
| | | | | |
G G A T – C G A

IV. APPROACH & METHODOLOGY

As stated in [16], one of the fundamental steps that we need to undertake to solve a problem in parallel is to split the computations to be performed into a set of tasks for concurrent execution. Decomposition techniques are broadly classified as recursive decomposition, data-decomposition, exploratory decomposition, and speculative decomposition. In most cases the problem on hand would determine which type of decomposition technique has to take place.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

TABLE I. THE DP MATRIX AND THE TRACE BACK PATH

	-	G	A	A	T	T	C	A	G	T	T	A
-	0	0	0	0	0	0	0	0	0	0	0	0
G	0	5	1	0	0	0	0	0	5	1	0	0
G	0	5	2	0	0	0	0	0	5	2	0	0
A	0	1	10	7	3	0	0	5	1	2	0	5
T	0	0	6	7	12	8	4	1	2	6	7	3
C	0	0	2	3	8	9	13	9	5	2	3	4
G	0	5	1	0	4	5	9	10	14	10	6	2
A	0	1	10	6	2	1	4	14	10	11	7	11

In sequence homologous search data-decomposition is used at two levels

1. Inter-process level: database is partitioned and distributed to different processes. This level uses MPI.
2. Intra-process level: the similarity matrix is partitioned and distributed among a set of threads. This level uses OpenMP.

Here we assume that each node will run only one process and one thread per core.

A. Coarse Grain Parallelism Using MPI

Pseudo code for the master node is shown in Figure 2. The master node has two tasks to handle. In the initial stage it will partition and distribute the database. This step will be performed using MPI. Afterword the master node will receive the optimal score from the worker nodes. At that stage the master node will start comparing results and then output the optimal result.

As aforementioned the master node is responsible for the data-decomposition process. In our approach the database sequences have approximately the same length of about 2000 characters. So we have chosen to statically partitioning the database sequences among the worker nodes.

Due to the static partitioning of the database the communication between the master node and worker nodes occurs only in two points at the start when the master node send the chunks to worker nodes and at the end when the worker nodes send the optimal score to the master node which in turn minimizes the communication overhead.

Figure 3 illustrates the communication between the master node and worker node. Only one message sent contains the database partition and one message received contains the optimal score.

```
Master Node
1. Read query sequence & scoring matrix.
2. Broadcast query sequence & scoring matrix to all
   workers.
3. Get database size
4. Statically partitioning database sequences in
   chunks = number of workers
5. For(i=1 to number of workers)
        Send chunk[i] to worker[i]
6. For(i=1 to number of workers)
        Receive optimal score from any worker
7. Combine and output the final result
```

Figure 2. Master Node Algorithm.

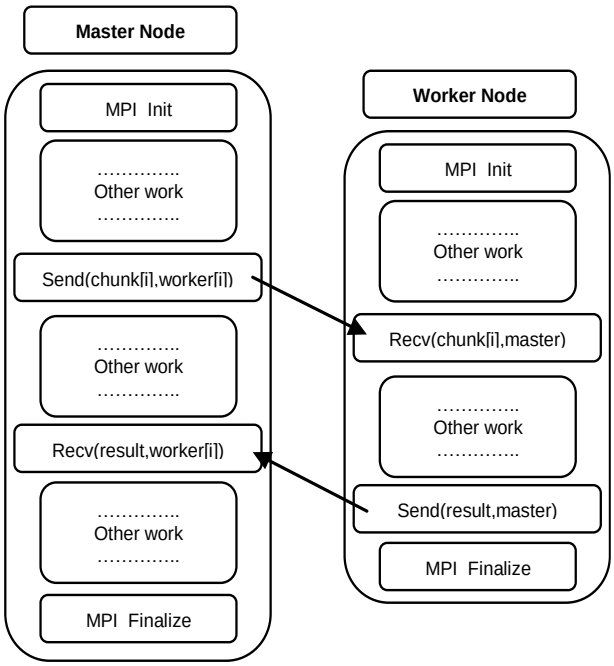

Figure 3. Master-Worker communication overhead.

B. Fine Grain Parallelism Using OpenMP

The challenge in implementing parallelism at the similarity matrix is data dependency. After the first row and first column has been initialized, the calculation of all other cells in the similarity matrix is dependent on the previous three cells. The previous three cell are one from the cell above (same column but previous row) one from cell on the left (same row but previous column) and one from the diagonal cell (previous row and previous column) as shown in Figure 4. Given the data dependency similarity matrix can be calculated anti-diagonal by anti-diagonal. To calculate any anti-diagonal we need only the two previous anti-diagonals so we can calculate the similarity matrix in parallel and linear space, an approach we chosen in our solution.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

As MPI used at the cluster level to send data to each worker, another level of parallelism is performed by the aid of OpenMP technology, where set of threads are forked to calculate the anti-diagonals. Main responsibilities of worker node are shown in Figure 5.

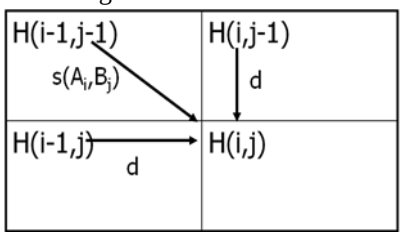

Figure 4. Data dependency in similarity matrix.

Worker Node

1. Receive query sequence & scoring matrix
2. Receive chunk[i]
3. For each target sequence in chunk[i]

   smith-waterman(query, target)

   Update optimal score

   END For
4. Send optimal score to master node

Figure 5. Worker Node Algorithm.

V. EXPERIMENTATION

The results introduced in this section are based upon the methodology discussed in the previous section. The smith-waterman algorithm has been implemented using MPI, OpenMP and Hybrid paradigm.

A. Experimentation Environment

The experiments were executed on a cluster consisting of five machines where one machine is the master node responsible for distributing data and four machines as worker nodes responsible of applying smith-waterman algorithm upon received data. All machines are of the same configuration, 2.2 GHz Intel core 2 Duo processor and 3 GB of RAM running ubuntu 10.04 (lucid) 32-bit operating system except the master node which is 2.2 GHz Intel quad core processor. The cluster is interconnected using Ethernet.

The MPI implementation used during experiments is MPICH2 which is prominent freely available portable MPI implementation. GNU GCC compiler implementation of OpenMP is used [17, 18].

The databases used during the experiments are a set of randomly generated DNA sequences from [19]. The size of the databases ranging from 500 kB up to 75 MB. The sequences' length ranging from 2000 to 2500 characters.

B. Experimentation Results

The main performance measure in our experiments is the relative speed up. Given two algorithms, where and are the execution times of the serial and parallel algorithms respectively. T2 varies according to the deployed parallel paradigm. MPI, OpenMP, and hybrid models are deployed such that the relative speed up S is calculated according to (2)

$$S = \frac{T_1}{T_2} \quad (2)$$

Figure 6 illustrates the execution time elapsed by the three parallel programming models, serial, MPI and hybrid model. Hybrid model which combines the MPI and OpenMP models gives better performance in terms of the execution time than the pure MPI model and obviously than the serial time. In this experiment 4 core 2 Duo CPUs are used as workers.

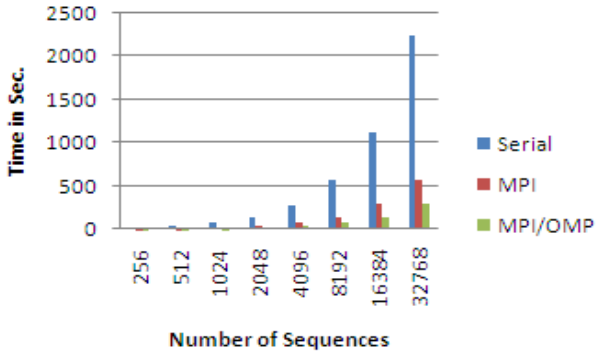

Figure 6. Execution time for three models on different database sizes with 4 CPU.

Figure 7 illustrates the speed up comparison between pure MPI and hybrid model. It is clear from the figure that the hybrid model provides better performance than the pure MPI model. From the experimental results we had noticed that the hybrid model can achieve a speed up of 7.497 on a cluster consisting of four dual core processors compared with only 3.84 speed up on the pure MPI implementation achieved at 32768 sequences. It is also clear that, the hybrid model performs better as compared to the pure MPI implementation as the sequence number increases.

Figure 8 illustrates the speed up comparison between the pure MPI model and the Hybrid model with different number of CPUs on the same database size of 32768 sequences. When running the experiment using only one worker, on pure MPI model the speed up was 0.8 which is obviously worse than the serial model. On hybrid model the speed up is raised to 1.52. It is clear that the speed up increases linearly as the number of CPU and cores increase.

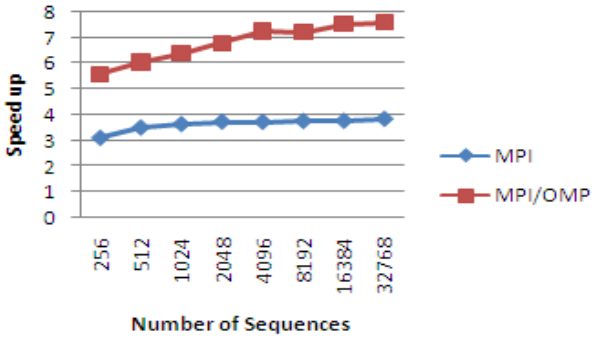

Figure 7. Speed up with 4 CPU.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

On our final experiment we compare the speed up of an MPI implementation with 4 worker nodes and an OpenMP implementation on one Node with 4 cores. Figure 9 illustrates the result. As mentioned previously, the communication between the master and workers occurs on two points only which decreases the communication overhead. The results indicate that the pure MPI is better than the pure OpenMP model in this case. This indicates that MPI is more scalable than OpenMP when applied to homology search problem even though combining OpenMP with MPI will increase the performance.

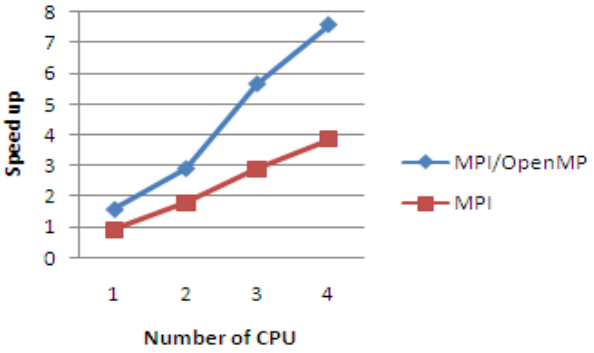

Figure 8. Speed up for different number of CPU on database of 32768 sequences

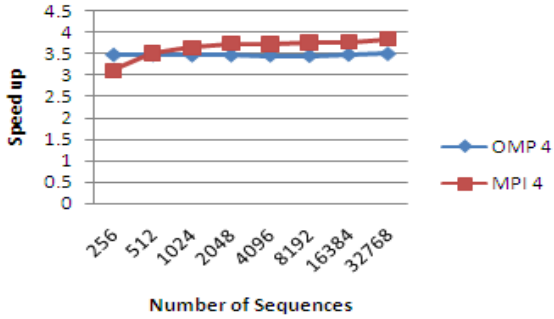

Figure 9. Comparison of 4 Cores and 4 CPU

REFERENCES

1. Kun-Mao Chao and Louxin Zhang, *Sequence Comparison: Theory and Methods*, Springer, 2012 pp. 35.
2. Michael Farrar, "Striped Smith--Waterman speeds database searches six times over other SIMD implementations," Bioinformatics, 2007, pp. 156-161, doi: 10.1093/bioinformatics/btl582.
3. VIPIN CHAUDHARY, FENG LIU, VIJAY MATTA, and LAURENCE T. YANG, "Parallel implementations of local Sequence alignment: hardware and software," Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies, Wiley Series on Parallel and Distributed Computing , 2006, pp. 234.
4. Hsien-Yu, L., Meng-Lai, Y., and Yi, C. "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.
5. T. Smith and M. Waterman., "Identification of common molecular subsequences," Journal of Molecular Biology, 1981, pp. 195–197.
6. Thomas Rauber and Gudula Rünger, *Parallel Programming: for Multicore and Cluster Systems*, springer, 2011, pp. 93.
7. L.A. Smith. "Mixed mode MPI / OpenMP programming," UK High-End Computing Technology Report, 2000.
8. www-unix.mcs.anl.gov/mpi/mpich2.
9. www.lam-mpi.org.
10. www.open-mpi.org.
11. Thomas Rauber and Gudula Rünger, *Parallel Programming: for Multicore and Cluster Systems*, springer, 2010, pp. 197.
12. The OpenMP API specification for parallel programming. http://openmp.org/wp/openmp-specifications/.
13. Georg Hager, Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2011, pp.143.
14. Drosinos, N., and Koziris, N. "Performance comparison of pure MPI vs hybrid MPI-OpenMP parallelization models on SMP clusters," 18th Int. Parallel & Distributed Symposium, 2004, pp.15.
15. http://www.clcbio.com/index.php?id=1046.
16. Ananth Grama, George Karypis, Vipin Kumar and Anshul Gupta, "Introduction to Parallel Computing, 2nd ed. Addison Wesley, 2003, pp.95.
17. http://www.mcs.anl.gov/research/projects/mpich2/.
18. http://gcc.gnu.org/.
19. http://www.bioinformatics.org/sms2/random_dna.html.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)