Indonesian Journal of Electrical Engineering and Computer Science
Vol. 9, No. 1, January 2018, pp. 234~242
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v9.i1.pp234-242

Parallelization of Pairwise Alignment and Neighbor-Joining Algorithm in Progressive Multiple Sequence Alignment

Agung Widyo Utomo, Wisnu Ananta Kusuma*, Sri Wahjuni
Department of Computer Science, Faculty of Mathematic and Natural Science
Bogor Agricultural University, Bogor, West Java, Indonesia

Article Info

Article history:

Received Sep 11, 2017
Revised Nov 27, 2017
Accepted Dec 10, 2017

Keywords:

ClustalW
Hybrid
Message passing
Multiple sequence alignment
Shared memory

ABSTRACT

A Progressive multiple sequence alignment ClustalW is a widely used heuristic method for computing multiple sequence alignment (MSA). It has three stages: distance matrix computation using pairwise alignment, guide tree reconstruction using neighbor-joining and progressive alignment. To accelerate computing for large data, the progressive MSA algorithm needs to be parallelized. This research aims to identify, decompose and implement the pairwise alignment and neighbor-joining in progressive MSA ClustalW using message passing, shared memory and hybrid programming model in the computer cluster. The experimental results obtained shared memory programming model as the best scenario implementation with speed up up to 12 times.

Corresponding Author:

Wisnu Ananta Kusuma,
Department of Computer Science, Faculty of Mathematic and Natural Science,
Bogor Agricultural University, Bogor, West Java, Indonesia.
Email: ananta@apps.ipb.ac.id

1. INTRODUCTION

Multiple Sequence Alignment (MSA) is a basic problem in Bioinformatics that aims to align more than two biological sequences to find similarity regions. MSA is used for phylogenetic analysis, identification of conserved motifs in a family of proteins and 3D homology modeling [1].

The most optimal pairwise alignment algorithm is dynamic programming, but when done on the MSA the complexity is O(LN) where L is the length of the sequence and N is the number of sequence that is the NP-Complete problem [1]. To overcome this problem, heuristic method is used, one of which is progressive MSA ClustalW [2] using guide tree as a guide in combining all pairwise alignment.

In addition to the complexity issue, the big data of sequence makes a challenge in applying MSA. On June 2017, it is known that there are 234 billion bases and 201 million sequences recorded in GenBank NCBI. This challenge makes the MSA algorithm needs to be parallelized to improve their performance. Some studies related to parallelization of MSA include ClustalW-MPI [3] which parallelize the first and third stages of ClustalW on 16 processors using MPI obtained speed up 4.3 times on progressive alignment. DIALIGN-TX [1] gained 3.13 times speed up using MPI and OpenMP hybrid systems on 28 cores heterogeneous multicore clusters [4] gets speed up three times using Star algorithm on MPI system with five computer units.

Implementation of parallel algorithms can be done partially to reduce overall computation time. This is done by [3] in parallelizing the first and third stages of ClustalW using MPI [5]-[6] only parallelize the neighbor-joining which is the second stage of ClustalW using the GPU [7] parallelize the pairwise alignment which is ClustalW's first stage using OpenMP.

This research aims to parallelize in the first and second stages of the progressive MSA ClustalW algorithm using the massage passing, shared memory, and hybrid programming model. The selection of the

first and second stages is based on complexity analysis and test results of ClustalW algorithm [5]. From the data presented by the research [5], it is known that the complexity and the computation time of the first stage (pairwise alignment) and the second stage (neighbor-joining) is greater than the complexity and the third stage computing time (progressive alignment).

The benefits of this research are expected to improve the performance of progressive MSA ClustalW algorithm, especially pairwise alignment and neighbor-joining so that it can support Bioinformatics research that requires MSA results.

2. RESEARCH METHOD

Explaining The method in this research adopted the steps in the decomposition of the program for parallelization [8]. The method is divided into four stages that can be seen in Figure 1.

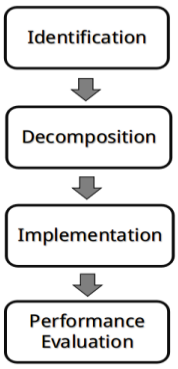

Figure 1. Research method

2.1. Identification

At this stage we identify the calculation components that can be run in parallel for the first and second stages of the progressive MSA ClustalW algorithm [8]. Identification is done by examining the pseudo code of both stages of the algorithm. The first stage of progressive MSA ClustalW is computing the distance matrix using pairwise alignment. In this research, the dynamic programming algorithm Needleman-Wunsch [9] is used as the optimal global alignment of the two sequences [10]. The algorithm has three stages: initialization, scoring matrix calculation, and traceback. The scoring matrix value is obtained by finding the maximum value of the cell on the left, the top and the diagonal of the intended cell. The distance value between these sequences is calculated using Equation 1. The distance values between the sequences are stored in the distance matrix.

$$The\ distance\ between\ sequences = 1 - \frac{Number\ of\ identical\ character}{Number\ of\ total\ alignment\ without\ gap} \quad (1)$$

The second stage is making a guide tree using neighbor-joining algorithm [8]. This algorithm uses distance matrix *D* from pairwise alignment as the input value, where *D(i,j)* is the distance between sequences/nodes i and j. Basically, this algorithm selects nodes *i* and *j* and merges them into new nodes in each iteration until leaving a single node [12]. Pairs *i* and *j* are selected by minimizing *Q(i,j)* as in Equation 2. While the value of *u(l)* is obtained according to Equation 3.

$$Q(i,j) = D(i,j) - u(i) - u(j) \quad (2)$$

$$u(l) = \sum_{k=0}^{r-1} D(l,k)/(r-2) \quad (3)$$

Notation *r* is the number of remaining nodes. The matrix will be updated by deleting the *i*-th row and the *j*-th column when the minimum *Q(i,j)* is found. New rows and columns are added to the matrix. The distance between the new node a and the remaining node *k* is described in Equation 4.

$$D(a,k) = \frac{1}{2}(D(i,k) + D(j,k) - D(i,j)) \quad (4)$$

2.2. Decomposition

At this stage, the identification results are decomposed and designed for implementation scenarios. Decomposition of parallel components is done by analyzing the potential of parallel whether it can be directly paralleled or require special techniques. Decomposition results are used to select the programming model for designing the implementation scenario.

Parallel programming model used is message passing, shared memory and hybrid. In message passing a computation consists of one or more processes that communicate with each other by sending and receiving messages [8]. Implementation of message passing is done on distributed memory system using Message Passing Interface (MPI). In the shared memory model, all processes share the same memory because it is done on one computer. Implementation of shared memory is done using Open Multi-Processing (OpenMP). The hybrid model combines the programming and implementation models of message passing and shared memory.

2.3. Implementation

The result of sequential pairwise alignment algorithm is verified using EMBOSS Needle from EBI UK and Global Alignment-BLAST from NCBI US. Both of them use Needleman-Wunsch algorithm. Verification is done twice using DNA sequences. The first verification uses *Ancylostoma duodenale mitochondrion* (NC_003415.1) and *Necator americanus mitochondrion* (NC_003416.2). The second verification uses *Human Papillomavirus type 132* (NC_014955.1) and *Human papillomavirus type 134* (NC_014956.1). The metric used to compare this algorithm is a similarity. The similarity is the percentage of the number of matching characters compared to the length of the alignment formed.

The result of the neighbor-joining algorithm is verified using NJ software from UQAM Canada and the guide tree of ClustalW software from DDBJ Japan. ClustalW is visualized using DrawTree from Phylogeny.fr France. Both of them use neighbor-joining algorithm. This Verification uses the same of data sequences in pairwise alignment verification. Verification is done using visualization data from neighbor-joining software output.

At this stage, the scenario is implemented in the computer cluster testing environment. In each scenario, both algorithms are run using a variation of 4 processors, 8 processors and 20 processors. As the input data are used randomly generated DNA sequences with an average sequence length of 600 bp. Each scenario runs on a variety of sequences of 20, 60, 180 and 540 pieces of DNA sequences. Testing is done as much as 10 repetitions for each scenario, variations of multiple cores are used, and variations in many sequences.

2.4. Performance Evaluation

The last stage is to compare the performance of parallel programs for each scenario. Performance comparison is done using speed up. speed up is the ratio of execution time to solve the problem in sequential to execution time to solve the same problem in parallel. A performance comparison is used to select the best parallel scenario. The pairwise alignment, neighbor-joining and combined algorithm are retested with three variations of sequence length and the number of sequences named LongSmall, AvgMedium, and ShortLarge as shown in Table 1.

Table 1. The Variation of Sequence Length and Number of Sequences

Attribute	LongSmall	AvgMedium	ShortLarge
Sequence Length	1400	350	90
Number of Sequences	100	300	700

2.5. Testing Environment

This research used a computer cluster consisting of one unit 8 cores, two units 4 cores and two units of 20 cores. The node that uses 8 cores is the head node. Each node uses an Intel® Xeon® CPU E5-2680 (2.70 GHz) processor with 32 GB RAM running on the Linux Ubuntu 14.04 LTS operating system. The speed of data transfer between nodes up to 10 Gb/s. The software used is OpenMPI 1.8.3 and GCC 4.8.2 compiler which supports OpenMP 3.1.

3. RESULTS AND ANALYSIS

3.1. Identification

3.1.1 Pairwise Alignment

Pseudocode of pairwise alignment ClustalW can be seen in Figure 2.

```
pairwiseAlignment(seq1,seq2)
1   m = seq1.length + 1
2   n = seq2.length + 1
3   for i = 0 to m
4     M[i,0] = i.GAP
5   for j = 0 to n
6     M[0,j] = j.GAP
7   for i = 1 to m
8     for j = 1 to n
9       diag = M[i-1,j-1] + similarity(seq1[i],seq2[j])
10      up   = M[i-1,j] + GAP
11      left = M[i,j-1] + GAP
12      M[i,j]= max(diag,up,left)
13  traceback(M[i,j])
14  distance = 1-(numMATCH/totalAlignmentWithoutGap)
15  Return distance

clustalW-pairwiseAlignment(N, seq)
1   for i = 1 to N
2    for j = 1 to N
3     If (i<j)
4     distMatrix[i,j]=pairwiseAlignment(seq[i],seq[j])
```

Figure 2. Pseudocode of pairwise alignment algorithm

The clustalW-pairwiseAlignment() function is an independent function used to set sequence combinations in pairwiseAlignment() function. The pairwiseAlignment() function returns the value of the distance between sequences such as Equation 1 that will be stored in the distance matrix. This function has data dependencies in scoring matrix computation (line number 7 to 12 in Figure 2) required cell data on the left, top and diagonal so when paralleled need the special technique to avoid a race condition. A race condition is a condition where two or more processes access shared memory at the same time, so the final result depends on the last executing process. This makes algorithms that have data dependencies more likely to be risky to parallelize.

3.1.2 Neighbor-Joining

Pseudocode of neighbor-joining can be seen in Figure 3. The neighbor-joining algorithm uses the distance matrix of pairwise alignment as the input value. The identification result of the pseudocode is known that the neighbor-joining algorithm has many loop constructs that have data dependencies. In Figure 3 line number 3 to 8 (representing Equation 3), the sum variable adds one cell line in the distance matrix and divides it by the remaining node to get the value of u on the i-th row. The processors will be simultaneously summing in parallel, so that the result is the last one summing up (being race condition).

Race conditions will also occur if the line number 9 to 15 and line number 17 to 22 are parallelized. This is because the processors search the minimum value in parallel so that resulting local minimum. Only the line number 25 and 26 (representing Equation 4) is the only independent parts of this algorithm. However, because the loop construction uses only one dimension, then the effect is not significant if compared to the previous parts.

```
neighborJoining(totalNode, distMatrix)
1  remainNode = totalNode
2  for remainNode down to 1
3    if remainNode > 3
4       for i = 0 to totalNode
5           sum = 0
6           for j = 0 to totalNode
7               sum += distMatrix[i,j]
8           u[i] = sum / (remainNode - 2)
9       for i = 0 to totalNode
10          for j = 0 to totalNode
11            Q[i,j]=distMatrix[i,j]-u[i]-u[j]
12            if Q[i,j] < minQ
13            minQ = Q[i,j]
14            node_i = i
15            node_j = j
16   else
17      for i = 0 to totalNode
18          for j = 0 to totalNode
19            if D[i,j] < minD
20            minD = D[i,j]
21            node_i = i
22            node_j = j
23
24    join node_i AND node_j
25    for k=0 to totalNode
26          distMatrix[a,k]=(distMatrix[node_i,k] +
            distMatrix[node_j,k] -
            distMatrix[node_i,node_j])/2;
```

Figure 3. Pseudocode of neighbor-joining algorithm

3.2. Decomposition

3.2.1. Pairwise Alignment

The result of identification indicates that clustalW-pairwiseAlignment() function can be parallelized without any problem because it has no data dependencies. Then in pairwiseAlignment() function needs a special technique to parallelize because it has data dependencies. Based on these two things will be made five implementation scenarios. The five scenarios are named pureMPI, pureOpenMP, ompInner, mpiOmpInner, and mpiOmpOuter.

The pureMPI scenario divides the outer loop task in the clustalW-pairwiseAlignment() function using MPI. Each task will be sent to a different processor by MPI, so the pairwiseAlignment() function for each pair combination can run in parallel. Data partitioning is done by dividing the number of sequences by the number of processors. The data have been divided is processed by each processor within each node, so that required inter-node communication.

The pureOpenMP scenario divides the outer loop task in the clustalW-pairwiseAlignment() function using OpenMP so that each pairwiseAlignment() function for each pair combination is executed parallel by the threads within a node. Partition data is done by dividing the number of sequences by the number of threads used.

The ompInner scenario parallels directly on the pairwiseAlignment() function using the shared memory programming model. The risk of data dependency is addressed by synchronizing [7]. The synchronization is done in the form of checking the cells required in the calculation. If the required cell value is available, then the thread can continue its work. But if the required cell is not available, then the thread will wait until that value is available. This data dependency causes this scenario to be slower than the previous independent scenario. The data partition used in this study uses a row-wise scheme which is one of the best schemes in implementing pairwise-Alignment() using OpenMP.

The mpiOmpInner hybrid scenario combines the pureMPI scenario with the ompInner scenario. The outer loop task in the clustalW-pairwiseAlignment() function is divided into the processor using MPI, then each processor performs the pairwiseAlignment() function that has been paralleled using OpenMP as the ompInner scenario.

The last scenario is mpiOmpOuter. It is a hybrid scenario that focuses on the outer loop of the clustalW-pairwiseAlignment() function. This scenario divides the number of sequences by the number of processors and distributes them to the compute node using MPI. The resulting partition data is further

subdivided by OpenMP into multiple threads. The end result is sent back into one to the head node using MPI.

3.2.2. Neighbor-Joining

The identification result obtained two parts algorithm which has a race condition problem and one independent part. Both parts can be handled with a parallel reduction that divides parallel processing into several steps resulting in a global solution. Implementation of parallel reduction can be done using OpenMP on the shared memory system. The selection of OpenMP as a neighbor-joining algorithm implementation is based on two things. First, the parallel reduction is more suitable for shared memory systems, and OpenMP has a parallel reduction feature that is easy to implement compared to MPI. Second, the structure of the neighbor-joining algorithm consists of many loop constructs which are the computational load of an algorithm. OpenMP is designed to parallelize loop construction without thinking about communication issues as MPI.

3.3. Implementation

3.3.1 Pairwise Alignment

Comparison of similarity values on the sequential pairwise alignment with both verification software can be seen in Table 2. Similarity of pairwise alignment obtained from both verification is not much different from EMBOSS Needle [14] and Global Alignment BLAST [15]. It shows the sequential pairwise alignment algorithm in this study has successfully verified with other software.

Table 2. Comparison of Similarity Values

Verification number	Pairwise Alignment	EMBOSS Needle	Global Alignment BLAST
1	83.7%	83.1%	82.7%
2	62.9%	56.8%	59.8%

Implementation of the five scenarios is done on 4, 8 and 20 processors. Each scenario has different configurations to use these three variations. The pureMPI scenario using 2 nodes × 2 processors, 2 nodes × 4 processors and 2 nodes × 10 processors for configuration. The pureOpenMP and ompInner scenarios use 4 threads on 4 cores node and eight threads on 8 cores node, 20 threads on 20 cores node. The mpiOmpInner and mpiOmpOuter scenarios use 2 nodes × 1 processor × 2 threads, 2 nodes × 2 processors × 2 threads and 2 nodes × 5 processors × 2 threads for configuration.

3.3.2. Neighbor-Joining

In this verification of neighbor-joining, *Ancylostoma duodenale mitochondrion* (NC_003415.1) is symbolized as seq1, *Necator americanus mitochondrion* (NC_003416.2) is symbolized as seq2, *Human papillomavirus type 132* (NC_014955.1) is symbolized as seq3 and *Human papillomavirus type 134* (NC_014956.1) is symbolized as seq4. Visualization tree from the calculation of neighbor-joining algorithm in this study can be seen in Figure 4. Visualization treefrom NJ software from UQAM Canada can be seen in Figure 5. Visualization tree from ClustalW from DDBJ Japan using DrawTree from Pyhlogeny.fr France can be seen in Figure 6.

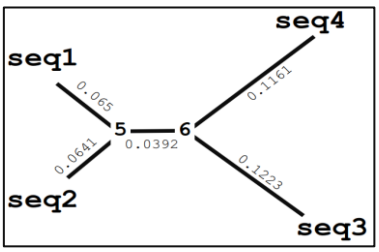

Figure 4. Visualization tree from neighbor-joining in this study

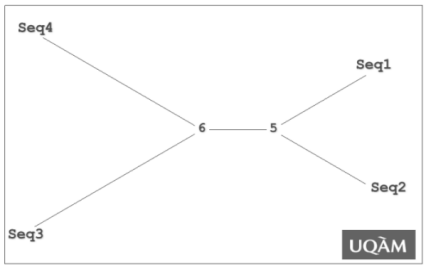

Figure 5. Visualization tree from NJ Software

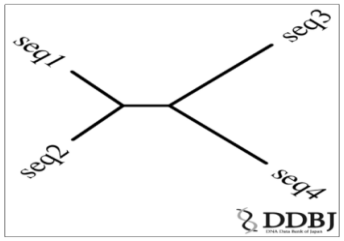

Figure 6. Visualization tree from ClustalW using DrawTree

From the three images can be seen that the visualization of the tree in different programs has the same shape. In addition to the tree shape, the position of seq1 with seq2 and seq3 with seq4 are in the same location. In Figure 4 and 5, it is known that node number five represents the new node from the merging of node seq1 and node seq2. These three points show the sequential neighbor-joining algorithm in this study has successfully verified with other software. Implementation of neighbor-joining is only done using OpenMP on the shared memory system. Implementation is done using 4 threads on 4 cores computer node and 8 threads on eight cores computer node and 20 threads on 20 cores computer nodes.

3.4. Performance Evaluation

3.4.1 Pairwise Alignment

At this stage, the five pairwise alignment implementation scenarios will be compared using the parallel program performance metrics to find the best scenarios. The implementation of parallel pairwise alignment algorithms has three dimensions of implementation variation: scenarios, number of sequences and number of processors. For ease of comparison, one variable will be chosen one variation so that the comparison becomes two dimensions.

In the use of 20 processor data as shown in Figure 7, it can be seen that the ompInner and mpiOmpInner scenarios can not complete the work when using 540 sequences. This is because both scenarios parallelize a part that has data dependency and it requires a lot of synchronization that overloads the operating system. In Figure 7 is also known that the pureOpenMP scenario has a much better speed up improvement than the other four scenarios. Based on that consideration, the pureOpenMP scenario is chosen as the best scenario.

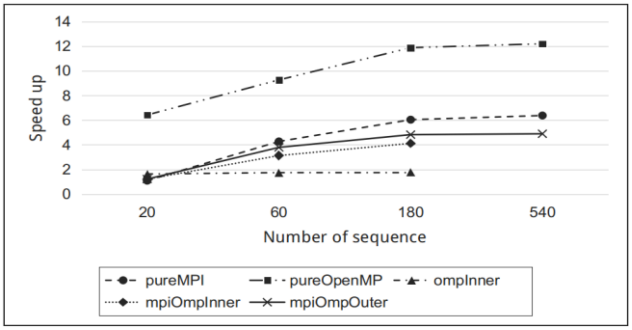

Figure 7. Comparison of speed up in parallel pairwise alignment using 20 cores computer

3.4.2. Neighbor-Joining

The result of the parallel neighbor-joining algorithm implementation can be seen in Figure 8. Improvement of speed up occurs when using 180 sequences. This happens because the load on each thread is too small in the number of sequences 20 and 60. The computation time becomes longer than the sequential implementation due to the cost of synchronization between threads in parallel reduction.

3.4.3. The Combined Algorithm

The pairwise alignment, neighbor-joining, and the combined algorithm of both tested using varying data to determine the characteristics and robustness of the algorithm. Figure 9 shows the comparison of speed up on three algorithms using LongSmall, AvgMedium and ShortLarge data. The graph shows that the combined algorithm is strongly influenced by the algorithm pairwise alignment with the speed up reaching 12 times. This is because the pairwise alignment algorithm has a larger portion of computation time compared to neighbor-joining. This does not mean that parallel neighbor-joining algorithm do not contribute to the combined algorithm. If the combined algorithm uses sequential neighbor-joining, then the speed up on the variation of ShortLarge will decrease to 4.7 times. Improvement of speed up in the neighbor-joining algorithm is achieved when the number of sequences is large. This contributes to keep robustness of the combined algorithm on a large number of sequences.

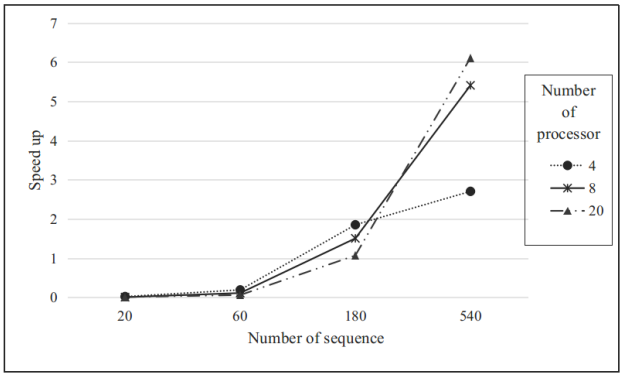

Figure 8. Comparison of speed up in parallel neighbor-joining

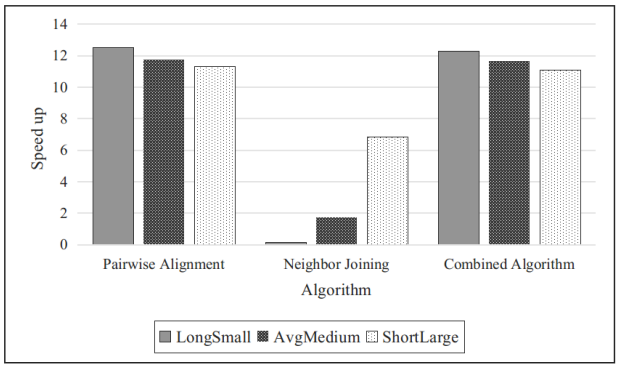

Figure 9. Comparison of speed up in pairwise alignment, neighbor-joining, and combined algorithm using variation of data

4. CONCLUSIONS

In this research, the best scenario is using a shared memory programming model with OpenMP implementation to parallelize the pairwise alignment and neighbor-joining algorithms. The combined algorithm using this scenario obtained speed-up up to 12 times on 20 computer cores. The combined algorithm proved robust in the variation of DNA sequences.

ACKNOWLEDGMENT

The authors would like to thank the Agency for Assessment and Application of Technology (BPPT) for the scholarship fund and the grid computing facility.

REFERENCES

[1] Macedo EA, Boukerche A. *Hybrid MPI/OpenMP strategy for biological multiple sequence alignment with DIALIGN-TX in heterogeneous multicore clusters*. IEEE International Parallel & Distributed Processing Symposium. Shanghai. 2011: 418-425.
[2] Thompson JD, Higgins DG, Gibson TJ. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.* 1994; 22(22): 4673-4680.
[3] Li KB. ClustalW-MPI: ClustalW analysis using parallel and distributed computing. *Bioinformatics*. 2003; 19: 1585–1586.
[4] Satra R, Kusuma WA, Sukoco H. Accelerating computation of DNA multiple sequence alignment in distributed environment. *Telkomnika Indonesian Journal of Electrical Engineering*. 2014; 12(12): 8278–8285.
[5] Liu Y, Schmidt B, Douglas LM. *Parallel reconstruction of neighbor-joining trees for large multiple sequence alignments using CUDA.* Proceedings of 23rd IEEE International Parallel & Distributed Processing Symposium. Rome. 2009.
[6] Zheng R, Zhang Q, Jin H, Shao Z, Feng X. A GPU-based parallel method for evolutionary tree construction. *Computers and Electrical Engineering*. 2014; 40(5): 1580–1591.
[7] Akbar AR, Sukoco H, Kusuma WA. Comparison of data partitioning schema of parallel pairwise alignment on shared memory system. *Telkomnika Indonesian Journal of Electrical Engineering.* 2015; 13(2): 694-702.
[8] Dongarra J, Foster I, Fox G, Gropp W, Kennedy K, Torczon L, White A. Source Book of Parallel Computing. San Francisco: Morgan Kaufmann Publishers. 2003.
[9] Needleman SB, Wunsch CD. A general method applicable to search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*. 1970; 48: 443-453.
[10] Utomo AW, Kusuma WA. *Penjajaran global sekuen DNA menggunakan algoritme Needleman – Wunsch.* Seminar Nasional dan Rapat Tahunan Bidang MIPA (Semirata 2014). Bogor. 2014; Book 3: 210-218.
[11] Saitou N, Nei M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*. 1987; 4(4): 406-425.
[12] Simonsen M, Mailund T, Pedersen CN. *Rapid Neighbour Joining*. Proceedings of 8th international workshop on algorithms in bioinformatics. Berlin. 2008.