

Graph Formations of Partial-Order Multiple-Sequence Alignments Using Nano- and Micro-Scale Reconfigurable Meshes¹

Mary M. Eshaghian-Wilner, Ling Lau, Shiva Navab, and David Shen

*Department of Electrical Engineering
University of California, Los Angeles
Los Angeles, CA 90095-1594
(310) 825-2647*

maryew@ee.ucla.edu, jonlau@ucla.edu, shiva_n@ee.ucla.edu, dshen727@ucla.edu

Abstract

In this paper, we show how to form Partial-Order Multiple-Sequence Alignment graphs on two types of reconfigurable mesh architectures. The first reconfigurable mesh is a standard micro-scale one that uses electrical interconnects, while the second one can be implemented at a nano-scale level and employs spin waves for interconnectivity. We consider graph formations for two cases. In one case, the number of distinct variables in the data sequences is constant. In the other case, it can be as much as $O(N)$. We show that given $O(N)$ aligned sequences of length L , we can combine the sequences to form a graph in $O(1)$ time, using either architecture if there is a constant number of distinct variables in the sequence. Otherwise, it will take $O(\log N)$ time if we use the spin-wave model and $O(N)$ time if we use the standard non-spin-wave-based version.

1. Introduction

Research in molecular biology has been moving at an astonishing pace in recent years. The rapid accumulation of biological data has necessitated more robust databases and data-processing algorithms. Consequently, the new area of computational biology is being created rapidly, combining the biological and informational sciences. To illustrate the complexities of some of the problems in computational biology, let us consider the genetic material in all living organisms, DNA. DNA is a polymer of nucleotides, where each nucleotide contains one of four bases: A, G, T, or C. An average gene in a human genome has 30,000 basepairs. With 30,000 genes estimated in each human genome, there are roughly 3 billion basepairs. A challenging task in computational biology deals with the finding of the “best” alignment amongst a given set of sequences. This is denoted as Multiple Sequence Alignment (MSA). The domain of MSA is not limited to DNA sequences. It could be for

proteins [3]; and it could become even more complex in sequence splicing problems [9].

The alignment of two sequences of length L can be solved via dynamic programming in $O(L^2)$ time. Extending it to N sequences, it will take $O(L^N)$ time, and in some cases the MSA problem is NP-complete [13]. Recent work includes CLUSTLW, T-COFFEE, etc. In one of the pioneering papers by Chris Lee, he demonstrated the advantages of applying a graph theoretical approach to the MSA problem (Partial Ordered Alignment); and many graph-based techniques have been proposed since then (see [7, 11, 13, 14] for details). Lee proposed the Partial-Ordered Multiple-Sequence Alignment Graph (PO-MSAG), and demonstrated its application in pair-wise alignment as well as its application in progressive pair-wise alignment [7, 11]. The results are comparable or even faster than some of the best-known algorithms.

In this paper, we study the complexity of the process to form a PO-MSAG. Assuming there are $O(N)$ sequences of length $O(L)$ that are already aligned, it will take at least $O(N*L)$ time to simplify the sequences sequentially into a PO-MSAG. Here, we present a set of fast and parallel algorithms to generate a PO-MSAG for any given set of aligned sequences at the hardware architecture level. The hardware can then be fabricated as a low-cost chip that can be used as a coprocessor with a more powerful processor (as shown in Figure 1).

We will consider two reconfigurable mesh architectures on which to implement our solution: the nano-scale spin-wave and the micro-scale electrical VLSI. We separate the problem into two cases: data sequences with a fixed amount of data variations and those with an arbitrary amount up to $O(N)$. In the fixed variations case, the spin-wave and the VLSI architectures can construct the PO-MSAG in $O(1)$ time. As for arbitrary $O(N)$ variations, the spin-wave architecture will construct the PO-MSAG in

¹ The authors are listed alphabetically by last name.

$O(\log N)$ time whereas the VLSI one takes $O(N)$ time. An example of the input sequence set is shown in Figure 2.

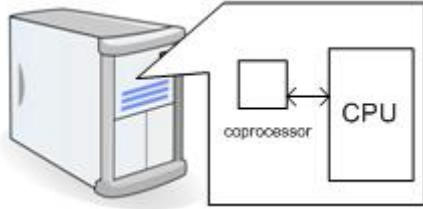


Figure 1: Coprocessor

```
A-C-G-T-T-A-C-T
A-G-T-T-G-G-G-C-T
(unsorted)
  ↓↓
A-C-G-T-T-A-*-*-C-T
A-*-G-T-T-G-G-G-C-T
(sorted)
```

Figure 2: Example of Partial-Order Multiple-Sequence Alignment

The rest of this paper will be organized as follows: section 2 gives some background information on reconfigurable meshes; section 3 shows how the two architectures can solve the MSA problem with a fixed amount of data variables; section 4 shows how to solve the problem with up to $O(N)$ data variables; and the final sections will discuss our concluding remarks and list our references.

2. Preliminaries

This section provides some basic information about Reconfigurable Meshes and our reasons for choosing this architecture.

2.1 Reconfigurable Mesh

A Mesh is a grid of processors, aligned in a two-dimensional array fashion. It is used as a model for massively parallel computing. The Reconfigurable Mesh is a special case of a Mesh, where connections between processors are adjustable. The reconfigurable mesh will be suited for mapping a PO-MSAG because of its ability to be reconfigured in the graph formation process. Each processor controls a local switch that can be reconfigured to control the physical connections between its four bi-directional bus lines [6]. Thus, we can control the flow of data to and from each processor with electrical routing [3]. For more information, please see [3], [5], and [12].

2.2 VLSI Reconfigurable Mesh Vs. Spin-wave Version

The VLSI Reconfigurable Mesh is a micro-scale architecture with standard reconfigurable-mesh properties (adaptable switches, bus channel, etc.) [6, 12]. The spin-wave reconfigurable mesh is a nano-scale spin-wave-based architecture. The spin-wave reconfigurable mesh is interconnected with ferromagnetic spin-wave buses. The difference between the two is that a spin-wave reconfigurable mesh allows simultaneous multiple transmissions and receptions on the same bus through different frequency channels [4]. This high level of connectivity between processor nodes can be valuable in solving the MSA problem. For more information about spin waves, please see [4].

3. PO-MSAG Formation for Constant Variation

The sequence output in this section is limited to at most a finite (constant) number of variables. To make the problem more concrete, we focus on DNA sequences, which have four variables (nucleotides) and possibly a special character, “*”, representing a blank space, as our example.

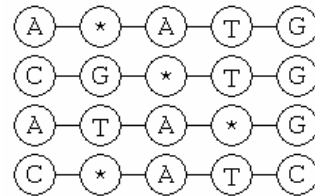


Figure 3: Example of a PO-MSA Output of DNA

3.1 Mapping on the Spin-Wave Architecture

1. INITIALIZE: In our first step, we map the given data sequences (length: L ; number of sequences: N) onto $N*L$ processors (we shall address these processors as nodes) each with its own memory capable of storing any 11 of the five possible data variables (in this case, the nucleotides: A, T, G, C, and the blank *) by placing each data variable of all sequences into the sixth memory slot of each respective processor. This takes constant time $O(1)$. So, for example, the first four nodes in rows and columns one and two will be mapped into memory as follows:

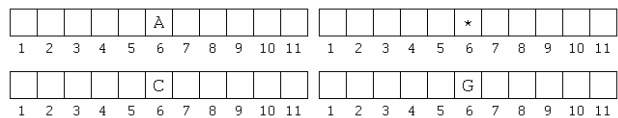


Figure 4: Example of the Memory Space of the Nodes

2. SET-UP: Next, we let each node visit its left neighbor and its right neighbor while storing these variables into the node's first and seventh memory slots, respectively. This step still takes constant time since retrieving the information is $O(1)$ and also since all nodes are performing this task simultaneously on parallel processors. So, once again, the first four nodes in rows and columns one and two will be mapped into memory as shown here:

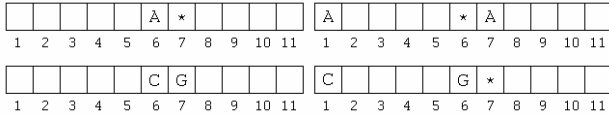


Figure 5: Memory Spaces of the Nodes After Setup Stage

3. ELIMINATION OF SIMILAR NODES:
 - a.) Now comes the grouping of the similar nodes in each column. We start by opening all the switches of each node so that all the channels between the nodes in the same column are open.
 - b.) Then, starting with the A nucleotide, then T, then G, then C, and finally *, we let each type of node simultaneously send out a signal identifying itself, and then turn off its switch, thereby closing that section of the channel. The closing of that particular section belonging to the node will still allow the node to receive signals, but it will not let signals travel beyond that node. The nodes that receive a signal will be the nodes with the same identity. Since we closed each section belonging to each node of the current round (i.e., all the A nodes), only one of them (the first node) will not receive the signal. We choose that first node to represent the rest of the group and eliminate the rest of the group. In other words, each of the A nodes will first send out an A signal and close off its section of the channel not allowing any signals to surpass that, while still being able to receive A signals itself. Then, there will be one – and only one – node left that has not received any A signals. The rest of the group will be obsolete, since the rest will now be represented by that first A node (all nodes that were linked to the other nodes will now link only to this node; all nodes that those nodes linked to will be linked to by this node). So, for example, just for the A nucleotide in the third column, this process will be as follows:

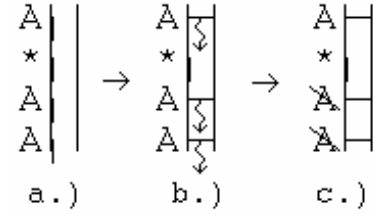


Figure 6: Disabling Nodes Using Spin Wave

- c.) We repeat the process for the T nucleotides, the G, the C, and finally for the *. This process also has a run time of $O(1)$ because in the spin-wave mesh, similar nodes can communicate to each other in constant time. Since we do this five times (with all A nodes, then T, G, C, and lastly *), the run time will be $5*O(1)$, which is still $O(1)$.
4. MEMORY UPDATE: Now, we must update the memory of the remaining nodes. We can do so by looking at the left and right neighbors of each of the remaining nodes (we will just take one A node as an example). If A's left neighbor's seventh through eleventh memory slots do not hold an A, the left neighbor must place an A node into its next available memory slot (eighth through eleventh). Also, if A's right neighbor's first through fifth memory slots do not have an A, then an A must be placed in the right neighbor's next available slot (second through fifth). These two procedures still run in constant time. That is because checking a node's left neighbor's memory for itself takes at most five tries (to check each of the left neighbor's five memory slots, seventh to eleventh). Placing the node in the next available slot of the left neighbor also takes five tries at most. The same number of tries applies to the node's right neighbor (10 tries total). Hence, this will all still be considered $O(1)$ time. At this point, each of the remaining nodes will have all of its left neighbors in its first through fifth memory slots and all of its right neighbors in slots seven through 11. We then repeat this process with the T, the G, the C, and finally the *. It still takes constant time because there are only five types of different nodes. With this kind of bookkeeping, we can easily form the PO-MSAG (usually by third-party graph-drawing algorithms) [8]. This concludes the implementation of our algorithm using the spin-wave technology. So the first two columns will look like this after the memory update:

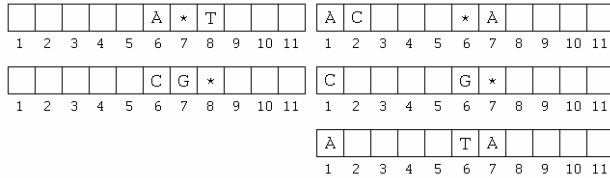


Figure 7: Memory Space after the Update Stage

3.2. Mapping on the Electrical VLSI Architecture

1. INITIALIZE: To implement our algorithm using electrical VLSI mesh, the initial mapping to memory of each data variable is identical to the spin-wave technique and will thus run in constant time, $O(1)$.
2. SET UP: Next we store each node's left and right neighbors in its first and seventh memory slots, respectively. This is still the same as in the spin-wave architecture, taking $O(1)$ time.
3. ELIMINATION OF SIMILAR NODES:
 - a.) Here is the main difference between spin-wave and VLSI meshes. In the electrical VLSI, only one signal can be sent through the wire at a time. In order for the nodes to communicate and group similar ones together eliminating repeats, we must send the signal and section off the wire, highly similar to our procedure with spin waves.
 - b.) We use an A node as our example. We first check to see if it is an A. If so, we send out a signal identifying the A and turn off its switch, which sections off the wire right after the node, allowing it to still receive signals but not letting the signal travel any further.
 - c.) Like previously, the node receiving no signal(s) will represent the rest of the similar nodes.
 - d.) We repeat again with the other variables (the T, G, and C nucleotides, and the blank *). This process still runs in constant time. For example, the process for the third column is as follows:

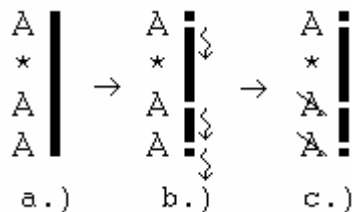


Figure 8: Disabling Nodes Using VLSI

4. MEMORY UPDATE: The last step of checking each node's neighbors and updating its memory is the same as the one shown in the spin-wave mesh and will take constant time. Thus, the overall runtime of this entire process using electric VLSI will also run in constant time, $O(1)$.

4. PO-MSAG Formation for $O(N)$ variables

The nature of the problem is the same (N sequences of length L on a reconfigurable mesh). However, the data set considered in this section can have as many as N different variations.

4.1. $O(N)$ Variables on the Spin-Wave Version

1. NODE ELIMINATION: Given an $N \times L$ Reconfigurable Mesh (RMESH), we will fill it with N sequences, each of length L , having arbitrary N data variables. In the following section, we will discuss the algorithm for eliminating repeated nodes in each column. The algorithm guarantees that there will remain only one of each kind of node.

Neighboring nodes connection: Every node will establish connections to its left and right neighbors and record their identities. This record will be extremely useful in the latter phase of the algorithm as we try to link the remaining nodes in the graph after the elimination.

Repeated node elimination: Repeated nodes will be eliminated in order to simplify the data set. To guarantee that only one copy of each kind of node remains, we encode or assign each node with a unique $\log N$ bit ID number starting from the top to the bottom of each column in the mesh. Then the algorithm below uses each node's unique ID to guarantee that only one of each kind remains.

Pseudo code for node elimination:

```

Loop
  Every node looks at its most significant bit (MSB)

  If MSB = 1
    Broadcast its 1's into the column
    channel
  If MSB = 0
    Listen to the column channel.
    If there is a 1 in the channel and a
    node's MSB is zero
      Disable
    End if
  End if
Until all log N bits are considered
  
```

Figure 9: Algorithm for Finding a Single Node to Replace the Rest of Its Kind in $O(\log N)$ Time

In a spin-wave RMESH, multiple data can be sent at the same time at multiple frequencies [4]. If we assign a channel to each type of node, then each type will be able to communicate freely to its own kind without interference to or from other nodes. For

example, assuming we have N types of nodes in one column, then we can assign a frequency channel to each type of node.

P_1	f_1
P_2	f_2
...	...
P_{N-1}	f_{N-1}
P_N	f_N

Table 1: Frequency Assignment Table

Once the nodes have their own communication channels, they can communicate freely amongst themselves. Therefore, each type of node can perform the elimination process simultaneously. As a result, despite the arbitrary number of variables (different nodes), we reduce the elimination time to $O(\log N)$ for all cases.

min	A	-	B	-	C	-	D	-	G	-	F	-	C
	G	-	B	-	K	-	E	-	K	-	G	-	C
	A	-	B	-	A	-	C	-	K	-	G	-	B
	A	-	B	-	D	-	D	-	F	-	G	-	T
max	A	-	B	-	C	-	D	-	E	-	H	-	S

Figure 10: RMESH After Elimination

2. **GRAPH RETRIEVAL:** At this point, all the repeated nodes have already been eliminated in the previous stages. This will simplify the data set and make it ready to be transformed into a PO-MSAG. Meanwhile, we also introduce a connectivity problem to the system. For example, in the third column of the figure above (Figure 10), nodes K, A, and D are supposed to be connected to B of the second column; but since some of the B nodes were eliminated, the connection of K, A, and D are now lost. To solve this problem, a technique to retrace the connectivity will be needed. We can actually split this problem into two sub-problems:
 - A. The topological connection of the node (i.e., which node does each node connect to)
 - B. The passing of data as we try to utilize the bus in the reconfigurable mesh

A. Topological Problem:

In this problem, we try to ensure the connectivity amongst the nodes as we extract the PO-MSAG representation. To show connectivity amongst all the nodes, we must rely on each node's neighbor information collected earlier, as well as the built-in column information.

After elimination, only one node of each kind will remain in each column. Although the original neighbor of each node might not be enabled anymore, a node of the same

kind will still be guaranteed to exist in the same column, just in a different row. Assuming that we have recorded the column number and neighbor information of each node, there will be enough information to make a partial-order graph. For example, consider the first three columns of a mesh with its repeated nodes eliminated (Figure 11). We know that the node G will have recorded B as its right neighbor.

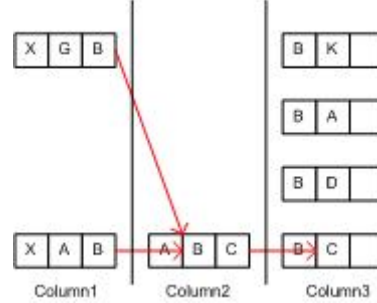


Figure 11: Example of Memory Space of the Nodes in the First Three Columns

After extracting the nodes, we need to map the connectivity. First we obtain half of the connections by focusing on each node's right neighbor memory. For instance, in column 1 (Figure 11), node G remembers that it is supposed to connect to node B (this information is recorded in G's right neighbor memory). To obtain the second half of the connection, we do a backward retrace.

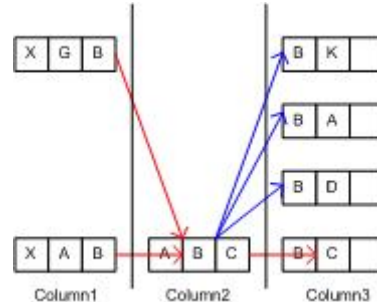


Figure 12: Backward Retrace

If a node does not have any forward connections, then it will look at its left neighbor and retrace its connection back to that neighbor. When both forward and backward tracings are complete, the nodes will be fully connected, assuming that the orders of the columns were preserved (i.e., column 1 always comes before column 2, while column 2 always comes before column 3).

B. Data Passing Problem:

If we know the connections amongst the nodes in the reconfigurable mesh, we will be able to pass messages effectively amongst them as well. We would also want to utilize the connection switches in a reconfigurable mesh. For example, assume that each node has two units: a reconfigurable transmitter unit and a reconfigurable receiver unit to transmit and receive, respectively, at

various frequencies. The way to solve this problem is by asking each node, "To whom should you listen?"

Each node will first tune its receiver to listen to the channel that belongs to its original left neighbor. Then it will tune its transmitter to transmit at its own frequency at the bus in the next column. Consider the example from before: in column 2, node B will reconfigure its transmitter to broadcast in its own frequency while nodes C and D will reconfigure their receivers to listen to the B channel. Whenever a package is routed from the node B, C and D will always be able to pick it up and forward it to the next level. Because every node within a column (all nodes are unique in the same column after elimination) can perform this algorithm simultaneously, the time complexity of this problem is $O(I)$.

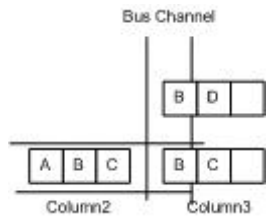


Figure 13: Communication between Nodes

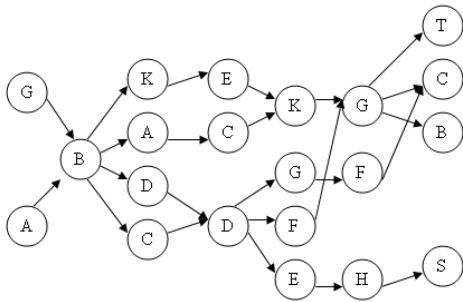


Figure 14: Result of the Example Sequence

1: Record Neighboring Nodes	$O(1)$
2: Node Elimination	$O(\log N)$
3: Enabling data forwarding	$O(1)$

Table 2: Time Analysis Table – Spin Wave

The above table lists the time complexities of each stage of the process. The total time complexity of the process will be $O(1) + O(\log N) + O(1) = O(\log N)$. The time for data forwarding could vary depending on the processes that might be applied to the graph. If we process the graph with labels, we need to ensure that the connection links amongst nodes are established. As for the formation of the graph, the forwarding of data (which runs in $O(N)$ time) has no effect on the time complexity. Before we conclude that this algorithm can lead to an optimal solution, let us compare another algorithm that will

perform the same function of forming a PO-MSAG on a reconfigurable mesh.

4.2. $O(N)$ Variables on the Electrical Version

The VSLI Reconfigurable Mesh (VMESH) shares many similarities with the spin-wave reconfigurable mesh we discussed in the previous section. The major difference is that in VMESH, multiple nodes cannot send messages at the same time. This poses major challenges when the application is communication-intensive. To draw a close comparison, we conform to the previous algorithm as closely as possible. The first step will eliminate the repeated nodes and the second step will reestablish connectivity.

1. **NODE ELIMINATION:** The conditions of the problem will be the same. The elimination process will again be handled via disabling the repeated nodes in each column.

Neighboring Nodes Connection: This step is exactly the same as in the spin-wave RMESH. Every node will look to the left and right neighbors and record their contents.

Repeated Node Elimination: We use the algorithm that was used in the constant variable cases, thus making the time complexity $O(N)$ in the worse-case scenario since there could be as many as N different variables (see section 3.2 for more details on the algorithm).

2. **GRAPH RETRIEVAL:** If we need to extract the nodes and map out the connectivity, we will do so with a similar method as before (see section 4.1) because the information about the neighbors is still preserved. For example, node G (Figure 11) will still have information about connecting to B even though the process is now performed on a VMESH. However, because the nodes can no longer communicate in different frequencies in a VMESH, a new way to handle data forwarding is needed.

We will be considering a bus structure to route data through the graph just as we did in the spin-wave case (sect 4.1). Except this time, nodes will no longer be able to communicate through their own channel, and communication will be limited by the number of nodes that are trying to communicate on the bus.

The nodes (assuming each has a transmitter and receiver) will reconfigure their receiver to listen only to the package if the sender is the same as its left neighbor. That means a $\log N$ bit pilot code is needed for each package sent in the bus. Since only one node can send at a time, it will take $O(N)$ time for the whole column to finish forwarding the nodes' data.

1: Record Neighboring Nodes	$O(I)$
2: Node Elimination	$O(N)$
3: Enabling data forwarding	$O(I)$

Table 3: Time Analysis Table – VLSI

The above table lists the time complexities of each stage of the process. The total time complexity of the process will be $O(I) + O(N) + O(I) = O(N)$. The time for data forwarding, $O(N)$, is not included because if we only want to form a PO-MSAG, data packages do not have to be routed through the RMESH. As for graph formation, we only care about the time it takes to set up the transceiver so that it is ready for data forwarding if needed.

To summarize the time complexities for both arbitrary variation and constant variation of the PO-MSAG formation problem solved with two reconfigurable mesh architectures is the table below.

	Arbitrary Variation	Constant Variation
VLSI	$O(N)$	$O(1)$
Spin Wave	$O(\log N)$	$O(1)$

Table 4: Summarized Time Analysis Table

5. Conclusion

In this paper, we demonstrated several techniques for forming graphs representing Partial-Order Multiple-Sequence Alignment of a given set of N -aligned sequences, using two types of reconfigurable mesh architectures (the spin-wave version [4] and the VLSI version [6, 12]). We showed that for a constant number of variables, the run times of both architectures are the same, $O(I)$. However for an arbitrary number of variables, the spin-wave architecture will have an $O(\log N)$ time complexity as opposed to an $O(N)$ time complexity using the other version. The algorithms described in this paper belong to one of the first sets of algorithms that we are currently studying in the area of bioinformatics. We intend to extend these results to large-scale graph databases. Furthermore, we intend to look into other applications that we believe will also benefit from such graphical representation of data, such as those in the areas of biological pathways and sequence splicing. Such areas will continue to demand even more efficient computing tools. We can expand the scale of our solution by using clusters of mainframes to aid in the sequence data processing. Mapping tools such as Cluster-M[5], can be used to handle the mapping and scheduling of graph data bases. All of these will serve as preliminary steps towards coming up with paradigms that could satisfy the computational needs of bioinformatics tasks.

6. References

- [1] Bromberg, Martin, "Partial-Order Alignment of RNA Structures," Undergraduate thesis, Brown University, RI, 2005.
- [2] Benjamin, Raphael, Degui Zhi, Haixu Tang, and Pavel Pevzner, "A Novel Method for Multiple Alignment of Sequences with Repeated and Shuffled Elements," *Genome Research* 14: 2336-2346, 2004.
- [3] Eshaghian-Wilner, Mary M., "Integrated Architectural Solutions for Protein Sequence-Structure Alignment," *Proceedings of the Sixth World Multi-Conference on Systemics, Cybernetics, and Informatics, SCI2002, Florida, July 2002.*
- [4] Eshaghian-Wilner, Mary M., Alex Khitun, Shiva Navab, and Kang Wang, "A Nano-Scale Reconfigurable Mesh with Spin Waves," *ACM International Conference on Computing Frontiers. Ischia, Italy, 2006.*
- [5] Eshaghian-Wilner, Mary M., "Mapping Arbitrary Heterogeneous Task Graphs onto Arbitrary Heterogeneous System Graphs," *International Journal on Foundation of Computer Science, Volume 12, Number 5, pages 599-628, 2001.*
- [6] Eshaghian-Wilner, Mary M., Russ Miller, "The Systolic Reconfigurable Mesh," *Journal of Parallel Processing Letters, Volume 14, Numbers 3 and 4, 337-350, 2004.*
- [7] Grasso, C., C. Lee, "Combining Partial Order Alignment and Progressive Multiple Sequence Alignment Increases Alignment Speed and Scalability to Very Large Alignment Problems," *Bioinformatics (Oxford, England), 20 (10): 1546-5, 2004.*
- [8] Grasso, C., M. Quist, M. Ke, C. Lee, "POAVIZ: a Partial Order Multiple Sequence Alignment Visualizer," *Bioinformatics (Oxford, England) 19 (11): 1446-8, 2003.*
- [9] Grasso, C., B. Modrek, Y. Xing, C. Lee, "Genome-Wide Detection of Alternative Splicing in Expressed Sequences Using Partial Order Multiple Sequence Alignment Graphs," *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing. World Scientific, 29-41, 2004.*
- [10] Lee, C., "Generating Consensus Sequences from Partial Order Multiple Sequence Alignment Graphs," *Bioinformatics (Oxford, England) 19 (8): 999-1008, 2003.*
- [11] Lee, C., C. Grasso, M.F. Sharlow, "Multiple Sequence Alignment Using Partial Order Graphs," *Bioinformatics (Oxford, England) 18 (3): 452-64, 2002.*
- [12] Miller, Russ, V. K. Prasanna-Kumar, Dionisios I. Reisis, and Quentin F. Stout, "Parallel Computations on Reconfigurable Meshes," *IEEE Transactions on Computers* 42 (6) 678 – 692, 1993.
- [13] Zhang, Xu, and Tamer Kahveci, "A New Approach for Alignment of Multiple Proteins," *Pacific Symposium on Biocomputing, Maui. 11:339-350, 2006.*
- [14] Yuzhen Ye, Adam Godzik, "Multiple Flexible Structure Alignment Using Partial Order Graphs," *Bioinformatics (Oxford, England) 21(10). 2362-2369, 2005.*