Parallel Multiple Sequence Alignment: An Overview

G. Scott Lloyd

March 25, 2010

Abstract

Multiple Sequence Alignment (MSA) is a fundamental analysis method used in bioinformatics and many comparative genomic applications. The time to compute an optimal MSA grows exponentially with respect to the number of sequences. Consequently, producing timely results on large problems requires more efficient algorithms and the use of parallel computing resources. In response to a growing volume of sequence data and the associated computational demand, parallel solutions to MSA are emerging. Some popular parallel system architectures and parallel MSA algorithms are presented along with examples of their performance. This overview discusses the most successful parallel methods and provides an in-depth review of core contributions in parallel MSA.

1 Introduction

Biologists and other researchers use multiple sequence alignment (MSA) as a fundamental analysis method to find similarities between nucleotide (DNA/RNA) and amino acid (protein) sequences. The computational cost for an optimal sequence alignment is on the order of $O(L^N)$ given L, the length of each sequence, and N, the number of sequences, which means that the exact solution to the problem is intractable for data sets with more than a few sequences. This complexity poses a challenge for sequence alignment programs to return results within a reasonable time period as biologists compare greater numbers of sequences. Consequently, many MSA approaches use heuristics to arrive at a sub-optimal solution in a reasonable time.

Even with current heuristic methods, a sequential alignment program may run for days or even weeks on a larger data set. Therefore, much research has been devoted to accelerating MSA with parallel methods. The large data sets now under analysis demand both efficient algorithms and high-performance parallel systems to produce timely results.

2 Parallel Systems

Several types of parallel systems have emerged to address computationally intensive problems. The most popular types will be introduced in this section. Hybrid systems that incorporate two or more of the following types are becoming more common.

Multiprocessor. Current multiprocessor systems usually consist of a cluster of nodes connected with a network. Each node typically has several processors or multi-core chips that share on-board memory. Memory on separate nodes is called distributed memory. Examples of these systems range from clusters of workstations to supercomputers with high-performance networks. Multiprocessor systems are classified as multiple-instruction, multiple-data (MIMD) systems.

Vector. A vector processor has extra instructions that operate on several data elements in parallel, thereby reducing the time needed to perform the same operation on several data elements. For example, a vector instruction may add all the numbers in one linear array to corresponding numbers in another. Using a scalar instruction, each element must be added individually and would require more time. Vector systems can be classified as single-instruction, multiple-data (SIMD) systems. Current x86-based processors have vector instructions in the form of Streaming SIMD Extensions (SSE).

Cell. Recently, the Cell Broadband Engine has received attention as an accelerator. However, further interest in the Cell may be diminished by the announcement from IBM to discontinue Cell production for technical computing [7]. The multi-core Cell contains one 64-bit PowerPC RISC-processor and eight 128-bit vector processors. The PowerPC processor manages the other processors and typically runs an operating system. Each of the vector processors have their own dedicated, local memory that is used for both code and data. Communication between processors occurs with DMA transfers.

GPU. Another popular acceleration technology is the general purpose graphics processing unit. Its commodity nature has sparked much interest outside of the graphics community as an acceleration engine. GPUs work in conjunction with a host system as a co-processor to offload compute-intensive kernels of code that can work somewhat independently in parallel on similar types of data. Current GPUs contain several hundred processors that are capable of floating-point and integer operations. Processors are arranged in groups and communicate through a complex hierarchy of memory systems.

FPGA. Field-programmable gate arrays (FPGAs) are used in reconfigurable computing systems as hardware accelerators. Like other accelerators, they act as co-processors and communicate with a host system through messages or shared memory. The hardware logic on FPGAs is configurable, which allows multiple processing elements with custom operations and data types to be programmed into the chip. These elements execute in parallel at hardware speed on data supplied from the host. A local interconnect that is designed in conjunction with the processing elements provides communication with very low overhead when compared with commodity processors.

3 Parallel Algorithms

Many MSA algorithms have been implemented on parallel hardware. The algorithms can be classified into several different categories depending on the underlying solution strategy.

Some common classifications include optimal, progressive, iterative, stochastic, and hybrid. Each of these will be discussed in more depth.

Parallel algorithms are often categorized by their granularity, which refers mainly to the frequency of communication between parallel tasks. A coarse-grained algorithm typically has fewer, larger tasks that communicate every second or less, while a fine-grained algorithm may have many, smaller tasks that communicate more frequently. The classification is not rigid, and some examples may not properly fit one category or the other.

This overview is organized first by the algorithm class and then by the system. Algorithm details for each of the classes and ways to parallelize various MSA methods will be described in each section. The examples discussed show a contribution in either overall performance or in the application of a parallel algorithm. Only pairwise alignment algorithms that are used in conjunction with MSA will be addressed. For parallel pairwise algorithms applied to database search, a dated reference compiled by Zomaya [35] is available.

3.1 Optimal

First, pairwise alignment [8] will be defined and then extended for multiple alignment. Pairwise alignment is defined separately to simplify the presentation and because it is often used as a sub-procedure in heuristic MSA algorithms. Following a more formal definition, some optimal parallel methods will be described.

3.1.1 Optimal Pairwise Algorithm

Given a pair of *sequences* $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$ of *length* $|A| = m$ and $|B| = n$ from the finite alphabet Σ, a *pairwise sequence alignment* is obtained by inserting *gap characters* "-" into A and B. The aligned sequences A' and B' from the extended alphabet $\Sigma' = \Sigma \cup \{\text{"-"}\}$ are of equal length such that $|A'| = |B'|$.

An alignment score provides a metric to assess the quality of an alignment and represents a measure of similarity between sequences. For pairwise alignment, it is the sum of similarity values for each pair of aligned characters. Characters that match have a positive value while those that mismatch have a lower or negative value. Any character aligned with a gap also contributes a negative value to the alignment score. In practice, the similarity value comes from a substitution matrix that reflects the probability of substituting one character for another. Let the function $s : \Sigma \times \Sigma \to \mathbb{Z}$ determine the similarity of two characters and let l denote the length of an alignment. The *pairwise scoring function* is then given by

$$F_{PW}(A', B') = \sum_{1 \leq i \leq l} s(a'_i, b'_i).$$

The goal is to find an optimal pairwise alignment of A and B such that for all possible alignments, the score is maximal. Pairwise alignment is typically solved with dynamic programming (DP), which fills a two-dimensional matrix with score values. Let H denote the DP matrix and the element $H[i, j]$ the similarity score of sequences $a_1a_2...a_i$ and $b_1b_2...b_j$. Let α represent the cost of inserting or deleting a gap. An optimal alignment is obtained by maximizing the score in each element of H. The values of H are determined by the following recurrence relations for $1 \leq i \leq m$ and $1 \leq j \leq n$:

$$\begin{aligned} H[0,0] &= 0, \\ H[i,0] &= H[i-1,0] + \alpha, \\ H[0,j] &= H[0,j-1] + \alpha, \\ H[i,j] &= \max \begin{cases} H[i-1,j-1] + s(a_i, b_j), \\ H[i-1,j] + \alpha, \\ H[i,j-1] + \alpha. \end{cases} \end{aligned}$$

The matrix fill occurs in a forward scan from upper left to lower right because of dependencies from neighboring elements. This dependency limits the amount of parallelism that is achievable in computing the matrix to the elements along the scan wavefront. Following a forward scan, traceback starts from a designated lower right position and follows a path to upper left, thereby determining the best alignment.

3.1.2 Optimal MSA Algorithm

The definition of a multiple sequence alignment is a generalization of pairwise alignment. Given an ordered set of sequences $S = \langle s_1, s_2, ...s_n \rangle$, a *multiple sequence alignment* (MSA) $A = \langle a_1, a_2, ...a_n \rangle$ is obtained by inserting *gap characters* "-" into s_i such that the *aligned sequences* $a_i \in A$ are of equal length with $|a_i| = k$.

To determine the quality of a MSA, a more complex scoring function than the one for pairwise alignment is needed. Various assumptions about the relationship between multiple sequences lead to several possible scoring methods. The weighted sum-of-pairs (WSP) method is popular among MSA programs. It assumes that sequences are related by an evolutionary tree and that sequence weights are derived from this tree. The WSP method calculates a total score from the weighted pairwise score of all sequences. Let $F_{WSP} : A \rightarrow \mathbb{Z}$ be a *WSP scoring function* for an MSA A such that

$$F_{WSP}(A) = \sum_{1 \leq i < j \leq n} w_{i,j} \sum_{1 \leq l \leq k} s(a_i[l], a_j[l])$$

where n is the number of sequences, k is the length of aligned sequences, $w_{i,j}$ is the weight given to a pair of sequences, and the function $s : \Sigma \times \Sigma \rightarrow \mathbb{Z}$ determines the similarity of symbol $a_i[l]$ with $a_j[l]$.

The choice of scoring method inherently affects the nature of the alignment algorithm. After choosing a scoring function, a suitable algorithm is determined to maximize the score and thereby produce an optimal alignment. More specifically, the *MSA problem* is to find an alignment A given a set of sequences S such that for all possible alignments of S, the score $F_{WSP}(A)$ is maximal. Several scoring methods and MSA algorithms have been proposed and are described in a thorough review by Gotoh [9].

The DP solution to pairwise alignment may be extended to multiple alignment with an N-dimensional scoring matrix where N is the number of sequences. However, because of exponential time and space scaling problems, optimal alignment algorithms like DP are limited to a small number of sequences.

3.1.3 Optimal Parallel Methods

Multiprocessor. Even though optimal MSA is challenged with intractability, researchers have tried to push the capabilities a little further with parallel implementations. Helal et al. [10] described a method to partition an N-dimensional scoring matrix among several processors using a representation based on Mathematics of Arrays (MoA). Waves of computation proceed through partitions in parallel following a hyper-surface. MoA manages the dependencies between partitions and the scheduling of waves. A relative speedup of about 40 is seen with 64 processors for six sequences with a length of approximately ten.

FPGA. Masuno et al. [19] used an FPGA to accelerate optimal MSA for up to five, short sequences. The N-dimensional scoring matrix is computed in two-dimensional slices by the FPGA. Compared with a 2 GHz Pentium 4, a speedup of 298 is demonstrated when aligning four sequences and a speedup of 104 is shown for five sequences. However, alignment performance degrades substantially with more than five sequences. The maximum sequence length that can be aligned is limited by the amount of external memory on the FPGA board. When aligning four sequences, the length limit is about 256, and for five sequences, the limit is about 64.

3.2 Progressive

The most common heuristic algorithm used to solve the MSA problem in practical time, both sequentially and in parallel, is progressive alignment. Other heuristic algorithms have been studied, but they generally provide poorer quality or suffer from greater computational cost with limited improvement in alignment quality [23]. Almost all of the parallel MSA examples referenced in this section are based on the popular ClustalW program [31]. Several reasons for this popularity exist. First of all, the method has been trusted by biologists for almost two decades with quality that is still comparable to more recent algorithms. The trust is gained with alignment results that are similar to biologists expectations. Even though newer methods may have better performance or quality, ClustalW has become a recognized benchmark standard. The algorithm provides fairly good alignments across a diverse range of sequence types. Also, the alignment algorithm is relatively fast, simple, and understandable. The source code is freely available and well supported, which provides broad access to biologists and allows researchers to experiment with algorithmic variations without starting from scratch. An outline of the progressive alignment algorithm will be given before describing the parallel methods.

3.2.1 Progressive Algorithm

Progressive algorithms successively perform pairwise alignment on the most similar sequences and groups of sequences, until all sequences are aligned. A progressive alignment is accomplished in three main stages.

Stage 1: All sequences are compared pairwise with each other and the score is stored in a similarity matrix.

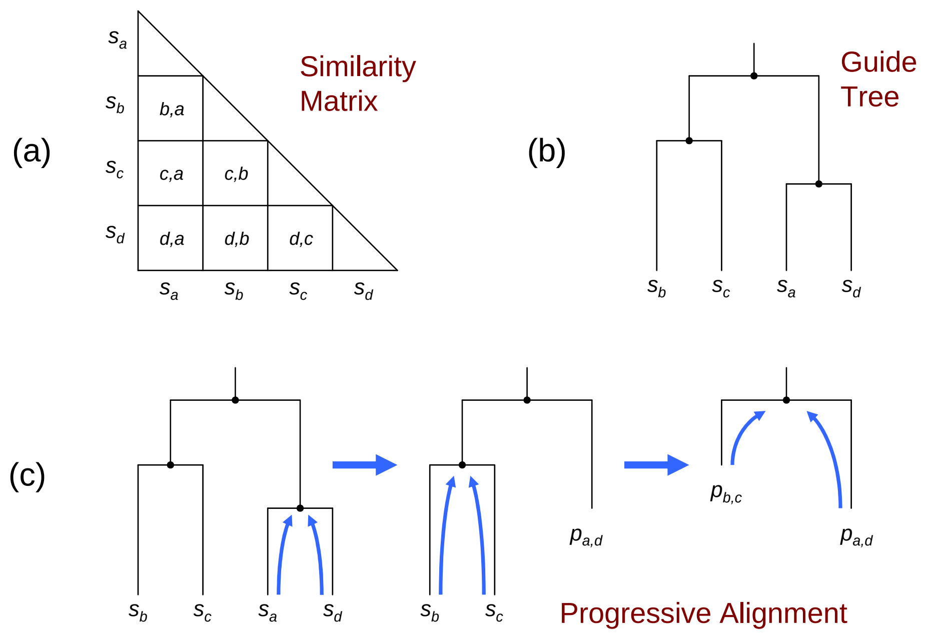

Figure 1: The three stages of progressive alignment: (a) compare all sequences to form a similarity matrix; (b) use the similarity scores and a clustering method to build a guide tree; and (c) progressively align sequences s_i and groups of sequences $p_{i,j,\ldots}$ in an order guided by the tree.

Table 1: Computational complexity of ClustalW stages given N, the number of sequences, and L, their typical length. An analysis is given by Edgar [5].

Step	Order (time)
Stage 1	$O(N^2L^2)$
Stage 2	$O(N^3)$
Stage 3	$O(N^3 + NL^2)$

Stage 2: A guide tree is constructed from the similarity matrix, with the leaves of the tree representing the sequences.

Stage 3: Following the branches of the guide tree from the leaves to the root, sequences and groups are pairwise aligned.

In the first stage, a comparison score is usually determined by counting of the number of identical characters in a pairwise alignment. Next, the second stage groups the most similar sequences together on terminal branches of a guide tree using a clustering method. This guide tree indicates the alignment order in the third stage where each node specifies a pairwise alignment (see Figure 1). Two groups of sequences are typically aligned with DP in a similar way to sequences; however, gaps inserted into groups occur a whole column at a time. The computational complexity of each stage is shown in Table 1.

3.2.2 Progressive Parallel Methods

Vector. The first published attempt to parallelize MSA was described by Tajima [28]. Only the DP calculations were vectorized on a FACOM VP-200 vector supercomputer. However, accelerating DP calculation with vector parallelism is problematic because vector machines typically do not include a table look-up vector operation that is needed to return the similarity score between two characters. To avoid a table look-up, the implementation by Tajima returns zero if symbols are equal and one otherwise. Parallelism was introduced with a FORTRAN 77/VP compiler that vectorizes DO loops when possible for a speedup of 4 on sequence-to-sequence alignment and a speedup of 2–3 on sequence-to-group alignment. A more recent attempt to use vector parallelism was reported by Chaichoompu and Kittitornkun [3] with only a speedup of 1.23 on ClustalW. The Intel C++ compiler was used with the /QxP option to generate Streaming SIMD Extensions (SSE) instructions for a Pentium processor.

Multiprocessor. Mikhailov et al. [20] parallelized all three stages of ClustalW on a shared-memory SGI Origin machine to demonstrate a speedup of 10 with 16 processors. The first stage is easily parallelized since each of the pairwise comparisons can be executed independently and the results stored in a similarity matrix without any conflict. Mikhailov introduced coarse-grained, task-level parallelism with OpenMP [1] directives. Since the first stage typically dominates the run time of ClustalW, most of the speedup comes from parallelizing this stage. The degree of parallelism with this method is limited to the number of all-to-all pairwise comparisons, which is $N(N-1)/2$.

A notable feature of Mikhailov's effort is the parallelization of the guide tree calculation in the second stage, whereas it is often overlooked in many other implementations because it requires the least computation time of all the stages. The clustering algorithm used in the second stage repeatedly finds a minimum element in the similarity matrix. Mikhailov's method searches each row of the matrix in parallel for the minimum element and then reduces the row-wise results to find the overall minimum. In this case, the parallelism is limited to the number of rows in the similarity matrix.

Since Mikhailov uses loop-level parallelism, only a portion of the available parallelism was realized in the third stage. During group-to-group alignment, ClustalW calls a function to determine the score for aligning two profile positions. A profile is derived from a group of sequences and consists of the character frequencies for each column in a group. A temporary matrix with dimensions equal to the length of each profile can be used to store the scores. Mikhailov's method precalculates each element of the scoring matrix in parallel.

Using message passing on a distributed-memory system, Li [13] also parallelized all the stages in ClustalW-MPI for an overall speedup of 14.6 on a 16 processor cluster. The test data consisted of 500 protein sequences with a length of 1100. Li used a fixed-size chunking strategy in the first stage to schedule 80 pairwise alignments to processors in a batch, thus reducing the frequency and overhead of communication. Li was the first to publish more sophisticated parallel methods for the third stage of progressive alignment. One method computes alignments at terminal nodes of the guide tree in parallel (see Figure 1). The problem with this method is that an unbalanced tree can severely limit the number of

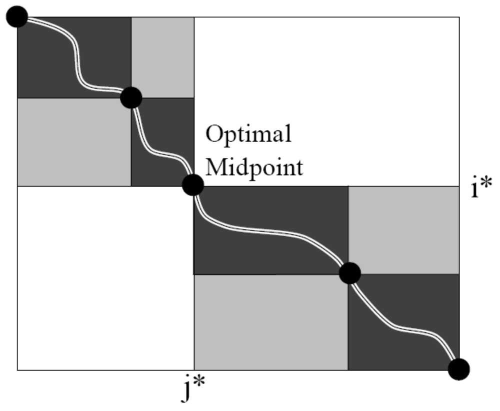

Figure 2: Myers and Miller subdivision algorithm [21].

parallel tasks. Furthermore, even a balanced tree will have limited parallelism near the root. In practice, the guide tree is usually unbalanced.

Another parallel method used by Li is based on the recursive Myers-Miller DP algorithm [21]. The Myers-Miller algorithm solves the pairwise alignment problem by dividing the DP matrix in half and then scanning from opposite corners towards the middle (see Figure 2). Where the two scans meet, an optimal midpoint in the traceback path is determined. This point becomes the corner of two subblocks which are in turn divided and scanned for midpoints. The recursion continues until a trivial alignment is encountered. The forward and backward scans can occur briefly in parallel, but must join before determining the midpoint. Each time a midpoint is found, two new subtasks can be spawned for the subblocks. Similar to guide-tree parallelism, recursive parallelism is also limited in the first steps. Using both guide-tree and recursive parallelism, Li only achieved a speedup of 4.3 on 16 processors in the third stage, while the first stage realized a customary linear speedup of 15.8.

The best performance reported for ClustalW using multiprocessors was by Tan et al. [29] with an overall speedup of 35 on an SMP-cluster system with 40 nodes and 80 processors. A speedup of 80 and 9.2 was obtained for the first and third stages respectively. In the third stage, Tan's method also distributes group-to-group alignments to system nodes using a method similar to Li that is based upon guide-tree and recursive parallelism. The main contribution comes from computing the forward and backward DP scans in parallel on processors within a node.

Load balancing strategies can improve the parallel efficiency in the third stage. Luo et al. [18] proposed a dynamic scheduling algorithm that estimates the execution time and communication cost for each task. Since the input for a node in the guide tree is dependent on a prior node, task costs are dynamically estimated after each task completes. The scheduler considers these costs and the current workload of the processors when making scheduling decisions. A peak efficiency of 0.75 is achieved with a speedup of 6 on 8 processors. In a later work, Tan et al. [30] also proposed a load balancing strategy based on tree accumulation. A speedup of 18 was achieved with 32 processors when aligning 3998 protein sequences.

Cell. Sachdeva et al. [26] ported the ClustalW application to the Cell platform for a Stage 1 speedup of 6.51 compared with a Xeon (Woodcrest) processor, but the overall performance was slower by a factor of 1.58. The significance of their effort comes from being the first to experiment with the Cell as a MSA accelerator and illuminating the challenges that must be overcome to achieve a performance improvement. Most of the speedup in the first stage comes from vectorizing the pairwise alignment computations and executing them in parallel on the Cell's eight Synergistic Processing Units (SPUs). However, vector performance is challenged on the SPUs with multiple branches in the DP code and also with table look-ups for the similarity score. The second and third stages were executed on the Cell's single, 64-bit Power Processing Unit (PPU). The lower performance of a PPU compared with a Xeon processor explains the overall performance degradation.

Vandierendonck et al. [32] accelerated ClustalW on two Cell BEs by a factor of 8 when compared with a 2.13 GHz Intel Core2 Duo processor running a single thread. Stage 1 is parallelized by vectorizing DP matrix calculations and scheduling the independent pairwise alignment tasks across the 16 available SPUs. Vandierendonck applied loop unrolling and loop skewing optimizations to compute the DP scan with diagonal vector operations. These optimizations are also applied to the group-to-group DP calculations in the third stage. Vandierendonck discovered that a significant portion of the third stage is spent calculating the similarity score between two profile positions. When comparing profile positions, all the character and gap frequencies must be considered. Similar to Mikhailov, these scores are precalculated in parallel, but in Vandierendonck's case, a more sophisticated scheme is proposed to pass precomputed scores from producer tasks through queues to consumer tasks. The PPU executes the sequential portions of ClustalW and load balances worker threads across SPUs with a dynamic scheduler.

Using a Playstation3, Wirawan et al. [33] achieved a peak speedup of 108 for the first stage when compared to a 3.0 GHz Pentium 4. The data set consisted of 1000 protein sequences with an average length of 446. Only the first stage was accelerated, and no overall speedup was reported. Wirawan used a sequence comparison algorithm that differs from ClustalW and has been previously demonstrated on FPGA [24] and GPU [15] accelerators. A count of matching characters is normally determined from an alignment, but in this algorithm the number of identical characters is computed directly by the recurrence relations during the forward scan of DP. By avoiding a full alignment, which requires a traceback procedure, better performance is realized.

GPU. Weiguo Liu et al. [15] were the first to publish MSA acceleration on GPUs and achieved a Stage 1 speedup of 11.7 compared with a 3.0 GHz Pentium 4 processor. Stages 2 and 3 were executed sequentially on the Pentium processor for an overall speedup of 7.2. A single GPU card (GeForce 7800 GTX) was programmed with OpenGL Shading Language (GLSL). The sequence comparison algorithm uses the same recurrence relations demonstrated on FPGA [24] and Cell [33] accelerators to assist in calculating the number of identical characters.

Yongchao Liu et al. [16] demonstrated an overall peak speedup of 41.53 on 1000 sequences of average length 858 with 1 GPU card (GeForce GTX 280) when compared with a 3.0 GHz Pentium 4. All three stages of ClustalW are accelerated by the GPU, with the parallel portions programmed using CUDA. When pairwise-alignment and guide-tree parallelism is low, cells of DP matrix calculations are computed in parallel. Since CUDA does not support recursion, a stack-based iterative version of the Myers-Miller algorithm was developed. This new version was used for both pairwise-alignments and group-to-group alignments. A separate paper [17] describes the parallel algorithm for the second stage. The neighbor-joining algorithm [27] is accelerated by computing the two innermost loops in parallel. Threads that compute minimum elements for square blocks of the distance matrix are scheduled on the GPU. The best speedup obtained in each of the three stages is 47.13, 11.08, and 5.9 respectively.

FPGA. Reconfigurable computing approaches accelerate the first stage of MSA by computing pairwise alignments with a pipeline of processing elements (PEs). This linear systolic array operates with fine-grained parallelism along a wavefront of cells in the DP matrix. The ClustalW algorithm does not use the score obtained from a pairwise alignment directly. Instead, the number of identical characters in an alignment are used to compute the fractional identity. Oliver et al. [25] accelerates the first stage of ClustalW, but leaves the second and third stages for execution on the host processor. Rather than actually aligning the sequences, custom hardware is developed to count the number of identical characters during the forward scan without performing traceback. The best overall speedup was 13.3 compared to ClustalW running on a 3.0 GHz Pentium 4. For Stage 1, a PCI-based accelerator board reached a peak speedup of 50.9 with 92 PEs in a Xilinx XC2V6000.

In another approach, Lin et al. [14] demonstrated an overall speedup of 34.6 using 10 Altera Stratix PEIS30 with a total of 3072 PEs. For the first stage, a speedup of 1697.5 was achieved when compared with a 2.8 GHz Xeon. The number of identical characters is deduced from the comparison score returned from the accelerator and the sequence lengths.

3.3 Progressive-Iterative Methods

Since progressive alignment is a greedy strategy, mistakes in placing gaps at early stages will remain throughout the process. To compensate for early mistakes, iterative refinement algorithms have been developed that repeat certain stages of the process a fixed number of times or until there is no improvement in the alignment quality. Group-to-group alignment is usually the most commonly iterated portion of progressive alignment. A more recent version of ClustalW now includes an iteration option to improve alignment quality, but this quality comes at the expense of more run time. To compensate for the lengthened run time, parallel methods have been introduce to some of the iterative applications. A few programs other than ClustalW have gained enough acceptance to warrant a parallelization effort.

MUSCLE. The iterative approach of MUSCLE starts with two rounds of basic progressive alignment and then repeats tree-guided group-to-group alignments until convergence is reached. As shown in Figure 3, a round consists of the three stages that are familiar to progressive alignment. The first two rounds derive pairwise similarity scores during Stage 1 in different ways, wherein the first round uses a faster alignment-free method based on k-mers and the second round uses the multiple alignment from the prior round.

Deng et al. [4] parallelized MUSCLE for a speedup of 15.2 on a 16 processor SMP system using OpenMP. The target data set consists of 50–150 proteins of average length 330. In

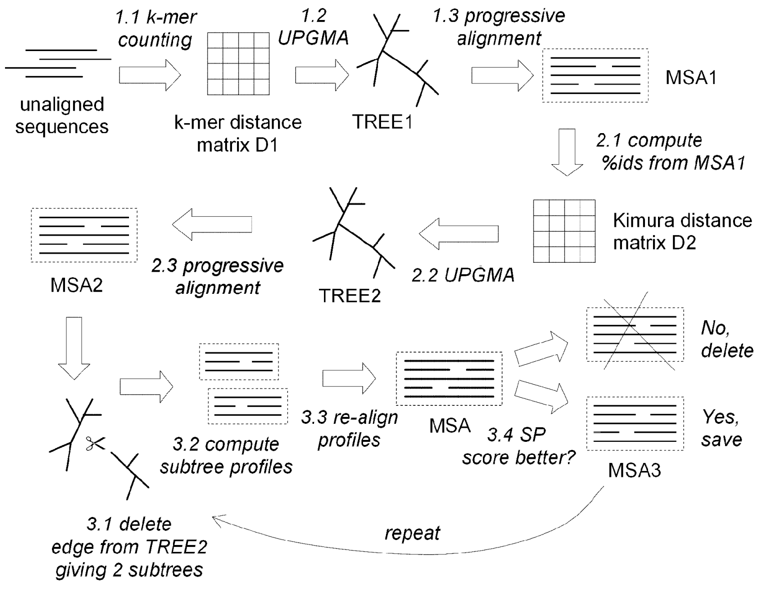
1.1 k-mer counting
1.2 UPGMA
1.3 progressive alignment
unaligned sequences
k-mer distance matrix D1
TREE1
MSA1
2.1 compute %ids from MSA1
Kimura distance matrix D2
2.2 UPGMA
TREE2
2.3 progressive alignment
MSA2
No, delete
Yes, save
3.2 compute subtree profiles
3.3 re-align profiles
MSA
3.4 SP score better?
MSA3
3.1 delete edge from TREE2 giving 2 subtrees
repeat

Figure 3: MUSCLE Algorithm [6]

the first and second rounds, group-to-group alignment following the guide tree is executed in parallel. A queuing module is used to schedule the tasks and make sure children nodes are aligned before parent nodes; however, as discussed before, this tree-based method has limited parallelism, and consequently, poor performance is reported for this stage with a speedup between 1 and 2. Most of the speedup comes from parallelizing and executing independently the all-to-all pairwise comparisons in the second round. Deng opted to use a more compute intensive probabilistic sequence comparison algorithm in the second round; therefore, most of the execution time was spent in this stage.

PRALINE. The progressive method of PRALINE has a pairwise sequence alignment stage and a progressive profile alignment stage that correspond to Stages 1 and 3, but the guide tree formation of Stage 2 is avoided. Instead of following a guide tree to align sequences and groups, PRALINE repeatedly chooses the next highest scoring pair to align until all sequences and groups are aligned to produce the final alignment. The highest scoring pair is determined by comparing all sequences with each other at first, and then comparing the aligned pair with the remaining sequences after each iteration.

A parallel implementation of PRALINE by Kleinjung et al. [12] realized a speedup of 10 with 25 processors on a distributed system using a set of 200 random sequences that are 200 residues in length. The pairwise sequence alignment stage is parallelized in the usual way by distributing pairwise alignments tasks to separate processors. In the progressive profile alignment stage, only the comparison of sequences and groups is parallelized. This occurs in a similar way to the first stage by distributing the comparison tasks, but each iteration must collect the results before selecting the highest score.

T-Coffee. While T-Coffee follows a progressive strategy, the first stage consists of a few extra steps that generate a library of pairwise alignments. This library is later used in the third stage to score alignments with a consistency-base objective function. After a round of basic progressive alignment, T-Coffee can iteratively refine the multiple alignment as an option. Each sequence is removed in turn from the multiple alignment and realigned with the remaining sequences.

Zola et al. [34] implemented a parallel version of T-Coffee using a master-worker architecture and message passing to obtain an overall speedup of about 40 on a system with 80 CPUs. Most of the parallelism comes from distributing pairwise alignment tasks with dynamic scheduling for a near linear speedup during library generation. In the progressive alignment stage, a sophisticated dynamic scheduling strategy is used that follows the guide tree, but almost no speedup is seen in this stage with more than 16 CPUs.

3.4 Stochastic Methods

Stochastic methods are used to solve optimization problems such as MSA. Two representative methods are simulated annealing and genetic algorithms. They are iterative in nature and produce a new prospective alignment as a random variation of a prior alignment. In each iteration, the prospective alignment is scored by an objective function to determine if it should be kept as the best found thus far. Since the alignments are prospective, many can be generated and evaluated in parallel. Creative methods are devised to efficiently update parallel tasks with better alignments that are found in other tasks at regular intervals. Since stochastic methods are generally much slower than other methods, they are often used to refine a seed alignment produced by a faster method.

Simulated Annealing. Ishikawa et al. [11] experimented with two parallel approaches to simulated annealing on a distributed system with 64 processing elements (PEs) connected by a 2-D mesh. The best approach, called temperature parallel SA, assigns each PE a fixed temperature for the duration of the simulation. After a fixed number of iterations, half the PEs exchange their best solution with one other PE that is close in temperature. The parallel algorithm reached a comparable scoring alignment in about one-fifth the time of the sequential simulated annealing algorithm. A conventional progressive algorithm was about twice as fast as the parallel algorithm; however, the parallel algorithm did eventually produce an alignment with a better score.

Genetic. Genetic algorithms are also known as evolutionary algorithms, since solutions or individuals evolve over time through mutation, crossover, and selection. Anbarasu et al. [2] implemented the island genetic algorithm in parallel using C and PVM on a distributed system. The method evolved five subpopulations of 20 individuals (alignments) in parallel, presumably on 5 processors. Better alignments are periodically sent to neighboring populations through a process called migration. Sending one migrant to a neighbor every 50 generations gave the best results. When aligning 15 protein sequences of length 292, the parallel algorithm ran ten times slower than ClustalW, but it achieved a slightly better score.

A more recent effort by Nguyen et al. [22], based on the Parallel Hybrid Genetic Algorithm, achieved a speed up of 6.6 on an eight processor cluster with the BAliBASE data set. Through a coarse-grained parallel approach, subpopulation evolution occurs on sepa-

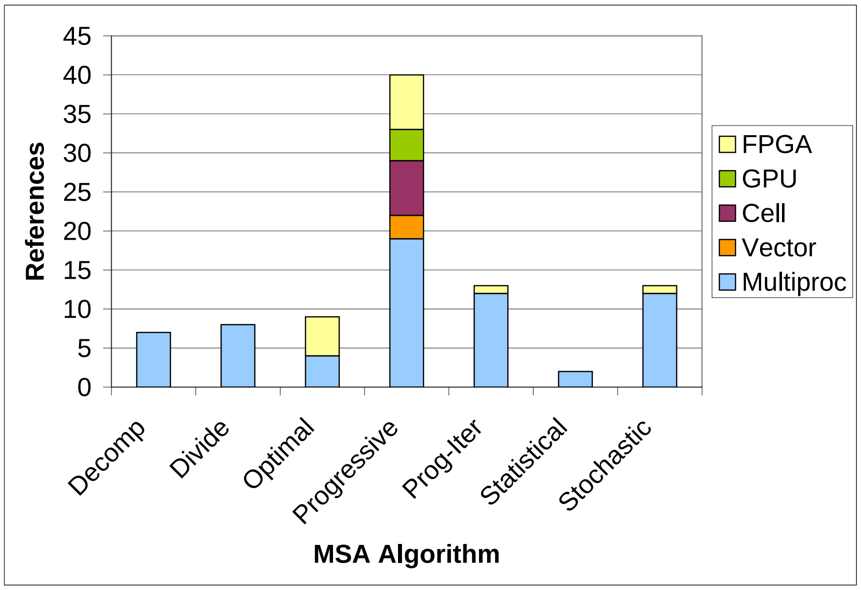

Figure 4: Parallel MSA references by algorithm

rate processors with periodic migration. The hybrid approach uses a progressive alignment algorithm to create the initial subpopulations and a genetic algorithm to refine them. Alignment quality was slightly better than ClustalW and a few other popular progressive-iterative alignment programs.

4 Discussion

Two parallel MSA approaches that divide or decompose the problem before using an underlying MSA method will not be discussed in this overview. These methods sit on top of other MSA methods (e.g. optimal, progressive) and typically reduce the alignment quality compared to the underlying alignment method in exchange for faster run times. Any of the discussed methods could be used as the underlying MSA method.

Most of the MSA algorithms implemented on parallel systems are based on a progressive strategy. Figure 4 shows the total number of parallel system references for several MSA algorithm classes and the different system types within each class. Progressive algorithms have the highest number of references and multiprocessor systems dominate the number of systems in all but the optimal class.

Compared with other MSA algorithms, progressive alignment algorithms have demonstrated the highest performance. The best known overall performance is reported by Yongchao Liu et al. [16] with an overall speedup of 41.53 for ClustalW on a GPU compared with a 3.0 GHz Pentium 4. While stochastic algorithms may offer more potential parallelism, they are slower by an order of magnitude or more and have not taken the lead on performance.

In the last few years, there has been a surge in MSA research using accelerator technology. Figure 5 shows the number of references for parallel MSA systems on a yearly basis.

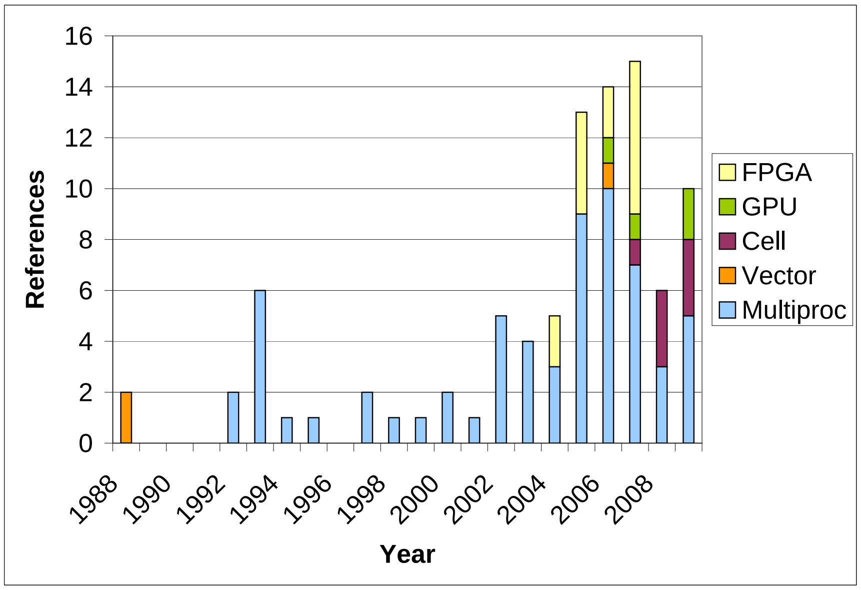

Figure 5: Parallel MSA references by year

The trend towards accelerator-based research started in 2005; however, research based only on multiprocessor systems appears to be waning. Since 2007, accelerator-based systems have accounted for at least half of the published parallel MSA research. Accelerator-based technology, whether existing or new, will likely play a significant role in future parallel MSA research.

References

[1] OpenMP application program interface, 2008.

[2] L. A. Anbarasu, P. Narayanasamy, and V. Sundararajan. Multiple sequence alignment using parallel genetic algorithms. In *Simulated Evolution and Learning*, volume LNCS 1585, pages 130–137. Springer Berlin / Heidelberg, 1999.

[3] Kridsadakorn Chaichoompu and Surin Kittitornkun. Multithreaded ClustalW with improved optimization for Intel multi-core processor. In *International Symposium on Communications and Information Technologies, ISCIT '06*, pages 590–594, 18-20 October 2006.

[4] Xi Deng, Eric Li, Jiulong Shan, and Wenguang Chen. Parallel implementation and performance characterization of MUSCLE. In *20th International Parallel and Distributed Processing Symposium, IPDPS 2006*, 2006.

[5] Robert C. Edgar. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(1):113, 2004.

[6] Robert C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[7] Michael Feldman. IBM cuts cell loose. http://www.hpcwire.com/blogs/IBM-Cuts-Cell-Loose-71994607.html, November 24, 2009. HPCwire.

[8] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, December 1982.

[9] Osamu Gotoh. Multiple sequence alignment: Algorithms and applications. *Advances in Biophysics*, 36:159–206, 1999.

[10] Manal Helal, Hossam El-Gindy, Lenore Mullin, and Bruno Gaëta. Parallelizing optimal multiple sequence alignment by dynamic programming. In *International Symposium on Parallel and Distributed Processing with Applications, ISPA '08*, pages 669–674, Sydney, Australia, December 10-12 2008.

[11] Masato Ishikawa, Tomoyuki Toya, Masaki Hoshida, Katsumi Nitta, Atushi Ogiwara, and Minoru Kanehisa. Multiple sequence alignment by parallel simulated annealing. *Comput. Appl. Biosci.*, 9(3):267–273, 1993.

[12] Jens Kleinjung, Nigel Douglas, and Jaap Heringa. Parallelized multiple alignment. *Bioinformatics*, 18(9):1270–1271, 2002.

[13] Kuo-Bin Li. ClustalW-MPI: ClustalW analysis using distributed and parallel computing. *Bioinformatics*, 19(12):1585–1586, 2003.

[14] Xu Lin, Zhang Peiheng, Bu Dongbo, Feng Shengzhong, and Sun Ninghui. To accelerate multiple sequence alignment using FPGAs. In *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)*, page 5, July 2005.

[15] Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig. GPU-ClustalW: Using graphics hardware to accelerate multiple sequence alignment. In *High Performance Computing, HiPC 2006*, volume LNCS 4297, pages 363–374. Springer Berlin / Heidelberg, 2006.

[16] Yongchao Liu, Bertil Schmidt, and Douglas L. Maskell. MSA-CUDA: Multiple sequence alignment on graphics processing units with CUDA. In *20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'09)*, pages 121–128. IEEE Computer Society, July 2009.

[17] Yongchao Liu, Bertil Schmidt, and Douglas L. Maskell. Parallel reconstruction of neighbor-joining trees for large multiple sequence alignments using CUDA. In *IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009*, pages 1–8, May 2009.

[18] Jiancong Luo, Ishfaq Ahmad, Munib Ahmed, and Raymond Paul. Parallel multiple sequence alignment with dynamic scheduling. In *International Conference on Information Technology: Coding and Computing, ITCC 2005*, volume 1, pages 8–13, April 2005.

[19] Shingo Masuno, Tsutomu Maruyama, Yoshiki Yamaguchi, and Akihiko Konagaya. Multiple sequence alignment based on dynamic programming using FPGA. *IEICE Transactions on Information and Systems*, E90-D(12):1939–1946, December 2007.

[20] Dmitri Mikhailov, Haruna Cofer, and Roberto Gomperts. Performance optimization of Clustal W: Parallel Clustal W, HT Clustal, and MULTICLUSTAL. SGI ChemBio, Silicon Graphics, Inc., 2001.

[21] Eugene W. Myers and Webb Miller. Optimal alignments in linear space. *Comput. Appl. Biosci.*, 4(1):11–17, 1988.

[22] Hung Dinh Nguyen, Kunihito Yamamori, Ikuo Yoshihara, and Moritoshi Yasunaga. Improved GA-based method for multiple protein sequence alignment. In *The 2003 Congress on Evolutionary Computation (CEC '03)*, volume 3, pages 1826–1832, December 2003.

[23] Cédric Notredame. Recent progresses in multiple sequence alignment: a survey. *Pharmacogenomics*, 3(1):131–144, 2002.

[24] Tim Oliver, Bertil Schmidt, Douglas Maskell, Darran Nathan, and Ralf Clemens. High-speed multiple sequence alignment on a reconfigurable platform. *International Journal of Bioinformatics Research and Applications (IJBRA)*, 2(4):394–406, 2006.

[25] Tim Oliver, Bertil Schmidt, Darran Nathan, Ralf Clemens, and Douglas Maskell. Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW. *Bioinformatics*, 21(16):3431–3432, 2005.

[26] Vipin Sachdeva, Michael Kistler, Evan Speight, and Tzy-Hwa Kathy Tzeng. Exploring the viability of the Cell Broadband Engine for bioinformatics applications. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2007*, pages 1–8, March 2007.

[27] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[28] Koji Tajima. Multiple DNA and protein sequence alignment on a workstation and a supercomputer. *Comput. Appl. Biosci.*, 4(4):467–471, 1988.

[29] Guangming Tan, Shengzhong Feng, and Ninghui Sun. Parallel multiple sequences alignment in SMP cluster. In *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)*, July 2005.

[30] Guangming Tan, Liu Peng, Shengzhong Feng, and Ninghui Sun. Load balancing and parallel multiple sequence alignment with tree accumulation. In *Euro-Par 2006 Parallel Processing*, volume LNCS 4128, pages 1138–1147. Springer Berlin / Heidelberg, 2006.

[31] Julie D. Thompson, Desmond G. Higgins, and Toby J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[32] Hans Vandierendonck, Sean Rul, and Koen De Bosschere. Accelerating multiple sequence alignment with the Cell BE processor. *The Computer Journal*, Advanced Access:inpress, 2009.

[33] Adrianto Wirawan, Bertil Schmidt, and Chee Keong Kwoh. Pairwise distance matrix computation for multiple sequence alignment on the Cell Broadband Engine. In *Computational Science – ICCS 2009*, volume LNCS 5544, pages 954–963. Springer-Verlag Berlin Heidelberg, 2009.

[34] Jaroslaw Zola, Xiao Yang, Adrian Rospondek, and Srinivas Aluru. Parallel T-Coffee: A parallel multiple sequence aligner. In *Proceedings of the ISCA 20th International Conference on Parallel and Distributed Computing Systems*, pages 248–253, September 24-26 2007.

[35] Albert Y. Zomaya, editor. *Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies.* John Wiley & Sons, April 2006.