



# Performance Optimization of Clustal W: Parallel Clustal W, HT Clustal, and MULTICLUSTAL

Dmitri Mikhailov, Haruna Cofer, and Roberto Gomperts

SGI ChemBio: [www.sgi.com/solutions/sciences/chembio](http://www.sgi.com/solutions/sciences/chembio)

## Introduction

Multiple sequence alignments represent a class of powerful bioinformatics tools with many uses in computational biology. Knowledge of multiple alignments (MA) helps to predict secondary and tertiary structures and to detect homologies between newly sequenced genes and existing gene (protein) families. With the adoption of high-throughput (HT) automation, offering scientists significantly more data with which to make better decisions, it is increasingly important to run MA calculations as quickly as possible to allow real-time decision making in the lab. The popular MA application Clustal W (1) provides a very good example of how the resources of multiprocessor shared memory computers can be utilized more efficiently. This paper first outlines the efforts undertaken by SGI to parallelize the Clustal W application. The parallel version shows speedups of up to 10x when running Clustal W on 16 CPUs and significantly reduces the time required for data analysis. Second, the development of a high-throughput version of Clustal W called HT Clustal and the different methods of scheduling multiple MA jobs in HT Clustal are discussed. Third, improvements in the recently introduced MULTICLUSTAL algorithm (2) and its efficient use of Parallel Clustal W are reviewed.

## Background

The basic idea behind the Clustal W algorithm for building MA is centered around aligning the most related sequences first. By aligning these sequences earlier on, one can minimize instances of unnecessary gap insertions. In order to identify the similar sequences the Clustal W algorithm estimates all pairwise combinations for the input protein sequences. For  $N$  input sequences one has to estimate  $N(N-1)/2$  pairs of sequences because of the pairwise matrix symmetry. This pairwise (PW) calculation (Fig. 1 left) represents the first stage of the algorithm. This is followed by the construction of a guide tree (GT), the second stage of the Clustal W algorithm. The guide tree contains the most closely matching sequences (or groups of sequences) located on the same branch (Fig. 1 middle). In this stage, a

default Neighbor Joining (NJ) method is utilized (3), the details of which are beyond the scope of this paper. Although the Clustal W algorithm offers other methods for tree calculation, NJ is known to be more robust and therefore is the only tree construction method optimized by SGI. The GT is further used as a guide during the third stage progressive alignment (PA), when the most related sequences are aligned first, followed by the alignment of more distant sequences or groups of sequences (profile alignment). The result of PA is the final MA, which can be output in a variety of plain text formats and viewed and analyzed with Clustal X or any other sequence viewer (Fig. 1 right).

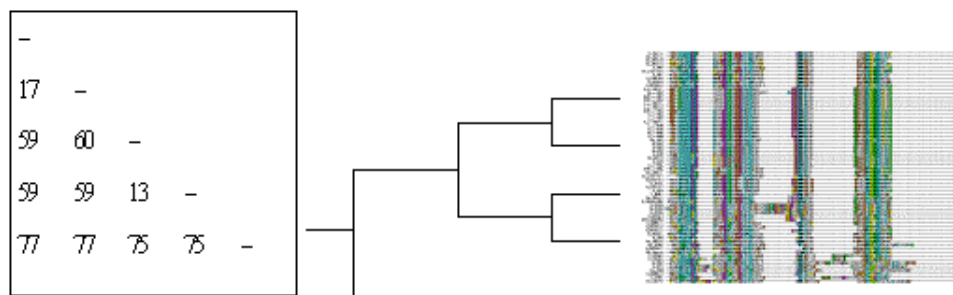


Fig. 1. Stages of the Clustal W algorithm. Left to right: pairwise calculation (PW), guide tree calculation (GT), and progressive alignment (PA).

## Parallel Code Optimizations

The Clustal W algorithm was parallelized by SGI using OpenMP<sup>TM</sup> (4) directives. The changes were added so that the parallel code is executed only if the user requests more than 1 CPU. Otherwise, the original serial code is executed.

Time profile analysis of the original Clustal W, using different numbers of G-protein coupled receptor (GPCR) proteins as inputs (Fig. 2), shows the following time fractions spent in each of the stages described in the previous section.

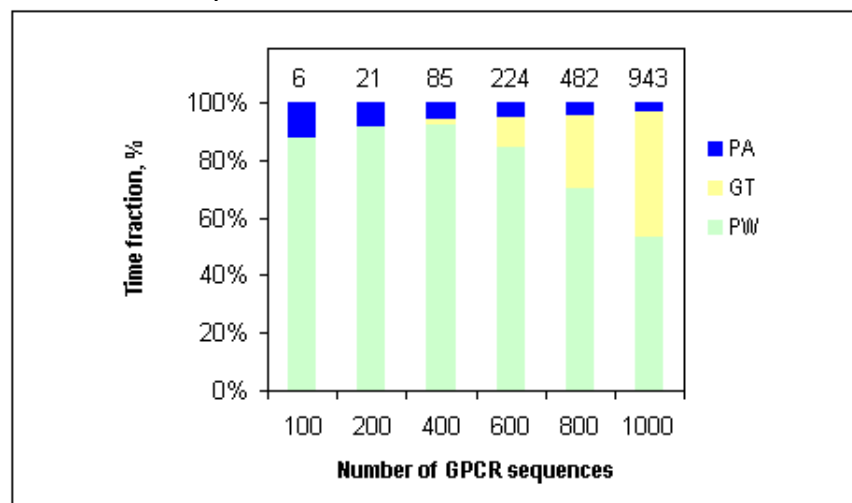


Fig. 2. Time fractions spent in three Clustal W stages for various input sizes. Total times in minutes are shown on the top of the bars.

Most of the time is spent in PW stage although the relative fraction is lower (~50%) for larger alignments

compared to ~90% for smaller alignments. Therefore, this stage needs to be parallelized first.

## PW Stage Optimization

As mentioned earlier, during the first stage  $N(N-1)/2$  pairwise comparisons have to be made. Because each comparison is independent of the rest, this part of the algorithm can be easily parallelized with the OpenMP "for" construct:

```
/* SGI pseudocode: Parallelize for pairwise distance matrix calculation */
#pragma omp parallel private(i,j)
{
#pragma omp for schedule(dynamic)
  for (i=Istart;i<Iend;i++)
  {
    for(j=i+1;j<Jend;j++)
      calculate_pairwise_matrix_element();
  }
} /* End of pragma parallel */
```

Because the size of the inner j-loop varies, the OpenMP "dynamic" schedule is used in order to avoid load unbalance among different threads. In principle the "static" interleave schedule can be used here as well, but because each pairwise comparison takes varying amounts of time, the "dynamic" type works better. This implementation is only efficient on a shared memory system like the SGI<sup>TM</sup> Origin 3000<sup>TM</sup> series. But no matter how well this stage is parallelized, the scaling would still be limited to a low number of processors if no further optimization is done. For example, without an additional parallelization, the alignment of 1,000 GPCR protein sequences, where PW stage accounts for ~50% (Fig 2) of total time, would be only ~1.8x faster when running on 8 CPUs according to Amdahl's Law<sup>5</sup>). In order to achieve better scaling for larger, more compute-intensive alignments, additional parallel optimizations are required for the second and third stages.

## GT Stage Optimization

In the second stage (GT calculation), the most time-consuming part is determining the smallest matrix element corresponding to the next tree branch. This can be done in parallel by calculating and saving the minimum element of each row concurrently and then using the saved minimum row elements to find the minimum element of the entire matrix:

```
/* SGI pseudocode: finding smallest matrix element */

min_row_element = (double *) ckalloc( (Iend - Istart) * sizeof (double));

#pragma omp parallel private(i,j)
{
#pragma omp for schedule(dynamic)
  for(i=Istart; i < Iend; i++) {
    for(j=Jstart; j < i; j++) {
      /* Compute smallest element of row i and store in min_row_element[i] */
    }
  }
} /* End of pragma parallel */

min_matrix_element = min_row_element[Istart];
```

```

for(i=Istart+1; i < Iend; i++)
    if(min_row_element[i] < min_matrix_element) {
        min_matrix_element = min_row_element[i];
    }

```

Note that the intermediate `min_row_element[ ]` array is created to store the smallest row elements in the parallel loop. The "dynamic" schedule is used here for the same reasons as in the PW stage.

## PA Stage Optimization

In the third stage, during the final PA, the original code makes calls to the `score(i, j)` function within a loop. Although the loop itself cannot be parallelized, the calculation of each `score(i, j)` can be precomputed in parallel. Specifically, a new matrix called `score_matrix[i, j]` is dynamically allocated and its elements are then calculated in parallel for all combinations of `i, j`:

```

/* SGI pseudocode: allocate and precalculate the score_matrix[] */
score_matrix = (int **) ckalloc((Iend - Istart) * sizeof (int *));
for(i=Istart; i<Iend; i++)
    score_matrix[i] = (int *) ckalloc ((Jend - Jstart) * sizeof (int *));

#pragma omp parallel private(i, j)
{
#pragma omp for schedule(static)
    for (i=Istart; i<Iend; i++)
        for (j=Jstart; j<Jend; j++)
            score_matrix[i, j] = score(i, j);
}/* end pragma parallel */

/*****
for(i=Istart; i<Iend; i++) {
    for(j=Jstart; j<Jend; j++) {
/*Original code*/
/*      hh = s + score(i, j); */
        hh = s + score_matrix[i, j];
    }
}

```

The loop is part of a function that recursively calls upon itself. As a consequence, unlike in the previous stages where parallel regions are entered only once, this parallel region is entered multiple times. Therefore, the "static" schedule is better suited here because it reduces OpenMP overhead.

The scaling of the parallel-optimized Clustal W is shown below for various-size inputs from the GPCR family (Fig. 3).

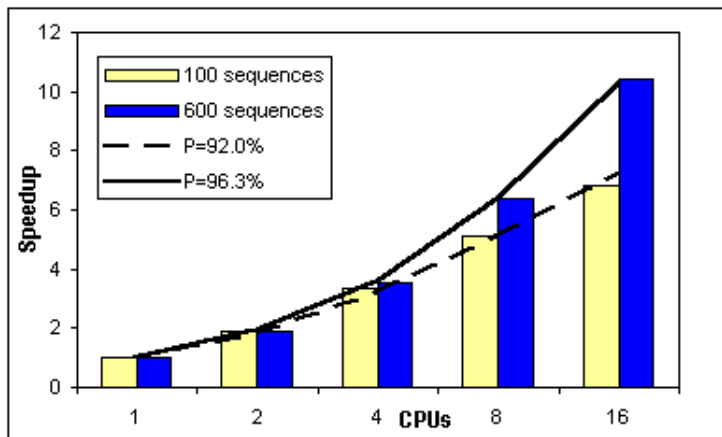


Fig. 3. Parallel Clustal W scaling. Calculation was done for 100 and 600 GPCR protein sequences with the average length 390 amino acids.

The relative scaling (in terms of fraction of parallel code P (5)) is better for larger inputs since most of the time is spent in the parallel first and second stages (Fig. 2). Parallelization of these stages is more coarse grained, and as a result the OpenMP overhead becomes minimal compared to the finer grained parallelization of the third stage. The speedup of more than 10x is seen for the MA of 600 GPCR proteins using 16 CPUs as compared to the uniprocessor time. Total time to solution is reduced from 1 hour, 7 minutes (uniprocessor) to just over 6.5 minutes (on 16 CPUs of the SGI Origin 3000 series), thus significantly increasing research productivity.

## Further Clustal W Improvements

### HT Clustal Optimization

The need to calculate large numbers of multiple alignments of various sizes is often seen in high-throughput (HT) research environments. To address this need, SGI has developed HT Clustal, a wrapper program that launches multiple Clustal W jobs on multiple processors, where each Clustal W job is usually a uniprocessor job. In order to reproduce this HT environment the following mix of heterogeneous MAs is constructed. In the example below, HT Clustal is used to calculate 100 different MAs for GPCR proteins (average length 390 amino acids). Each input file contains between 10 and 100 sequences taken randomly from a pool of 1,000 GPCR sequences. The number of sequences conforms to a gaussian distribution with the average of 60 sequences and standard deviation of 20 (Fig. 4).

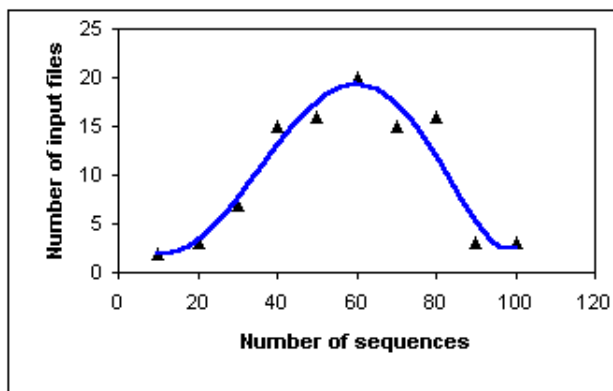


Fig. 4. Distribution of multiple alignment input files used for HT Clustal. The profile was constructed as described in the text.

In order to optimize the throughput performance, the input sequences are presorted based on a relative file size. The purpose of the presorting is to minimize load unbalance and hence improve the HT Clustal scaling. This load balance problem is clear on Fig. 5 (right) with horizontal bars representing individual Clustal W jobs submitted to each of 32 CPUs and is minimal with presorting (Fig. 5 left).

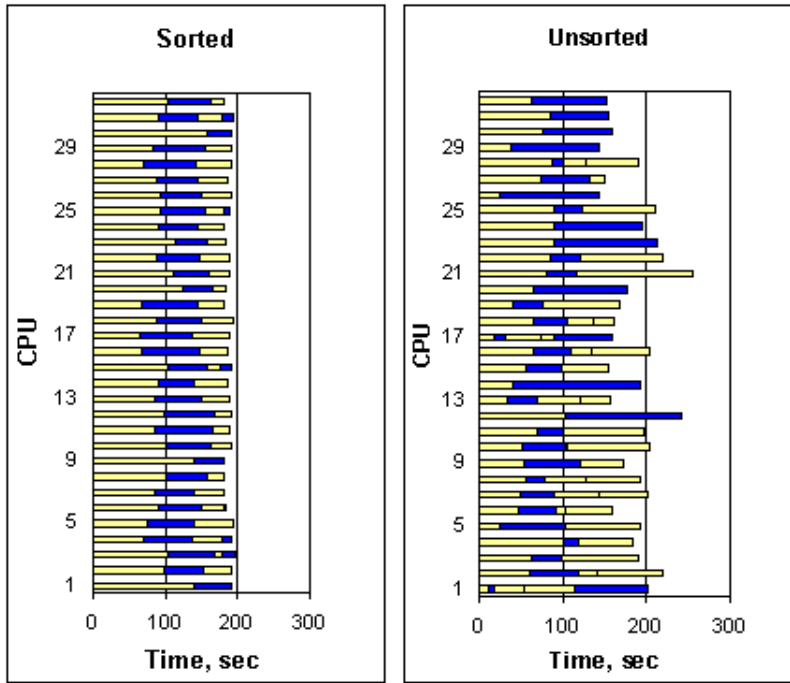


Fig. 5. Effects of input file presorting on load balance with a low number of jobs per CPU (100/32, or approximately 3). Each colored bar segment represents one uniprocessor Clustal W job out of 100 MAs used as the input for HT Clustal.

Empirical studies show that improvement from presorting becomes significant when the average number of jobs per CPU is on the order of 5. This is seen on Fig. 6 for the runs on 16 and 32 CPUs, where these numbers are  $100/16 \approx 6$  and  $100/32 \approx 3$ , respectively. When the average number of jobs per CPU is greater than 5, as seen for the runs on 1 to 8 CPUs (Fig. 6), statistical averaging reduces the load unbalance and there is only minor improvement with presorting.

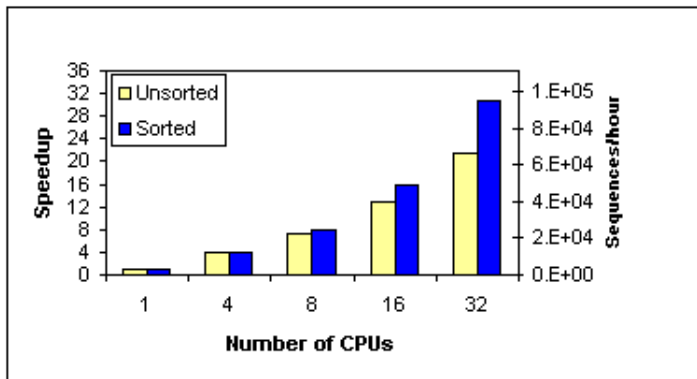


Fig. 6. HT Clustal scaling for unsorted and sorted input files. Throughput rate is measured in aligned sequences/hour and is shown on the right. The input described on Fig. 4 is used here.

With presorting one can achieve almost linear speedups. In the current example the speedups of 31x were achieved on 32 CPUs. For the larger test case (6) speedup of 116x was found on a 128-CPU SGI

Origin 3000 series server, thereby reducing total time to solution from over 18.5 hours to just under 9.5 minutes.

Because individual Clustal W jobs are uniprocessor, HT Clustal can be used efficiently on both single system image SGI Origin 3000 series servers and distributed Linux® clusters.

### MULTICLUSTAL optimization

The MULTICLUSTAL algorithm was introduced as an optimization of the Clustal W MA (2). Shown are the MAs for the sequences from the SH3 protein family (Fig. 7). The MULTICLUSTAL alignment (Fig. 7 right) shows domain structure more consistent with the 3D structures of proteins involved in this alignment.

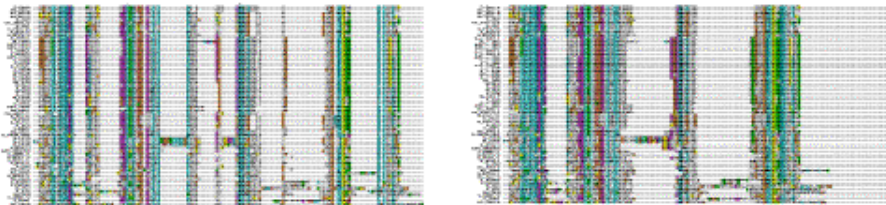


Fig. 7. SH3 proteins multiple alignments with default Clustal W parameters (left) and MULTICLUSTAL optimized (right). MULTICLUSTAL domain clustering is in better agreement with the SH3 protein 3D structures.

The algorithm searches for the best combination of Clustal W input parameters in order to produce more meaningful multiple sequence alignments (i.e. smaller number of gaps with more clustering). It does so by performing Clustal W calculations for various scoring matrices and gap penalties in the PW/GT and PA stages (Fig. 8).

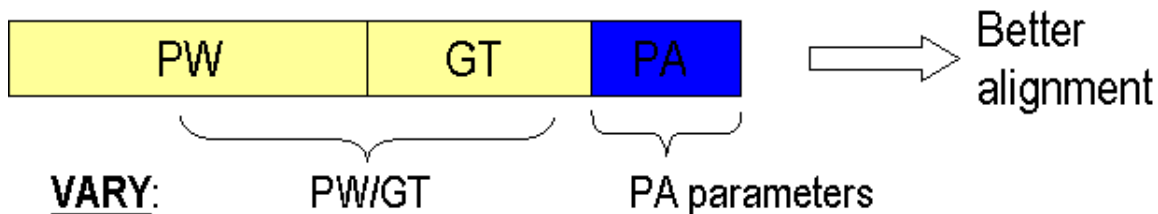


Fig. 8. MULTICLUSTAL algorithm independently varies Clustal W parameters for PW/GT and PA stages trying to find the most optimal MA.

SGI has optimized the original MULTICLUSTAL algorithm by reusing the tree calculated in the PW/GT step. Therefore, the guide tree is calculated only once for a given combination of PW parameters and is then used for all possible combinations of PA parameters. The performance (relative to that of original MULTICLUSTAL on 1 CPU) is shown in Fig. 9 and speedups range from 1.5 to 3.0 compared to the original algorithm running on the same number of processors.

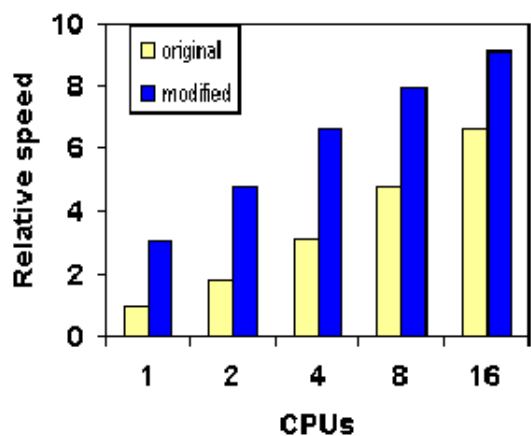


Fig. 9. Scaling of original and SGI modified MULTICLUSTAL.

Similar time to solution can be obtained using the SGI modified MULTICLUSTAL on smaller number of CPUs compared to the original MULTICLUSTAL, therefore freeing additional computer resources without a performance degradation.

## Conclusions

This paper shows how to optimize turnaround (Parallel Clustal W) and throughput (HT Clustal) performance of the Clustal W (1) and improve MULTICLUSTAL (2) algorithm. This work is part of the ongoing efforts within the SGI ChemBio group that also involve optimization of new and existing bioinformatics and computational chemistry algorithms (6), maintaining the latest software versions and user support.

## Availability

Original Clustal W and DBClustal source codes are available for download from:  
<ftp://ftp-igbmc.u-strasburg.fr/pub>

Parallel Clustal W (current version 1.81) and HT Clustal with setup instructions are freely available for download from:  
[www.sgi.com/solutions/sciences/chembio](http://www.sgi.com/solutions/sciences/chembio)

Users are requested to fill in general information, which is used for future release notices. At the date this article was written there have been over 900 registered downloads of Parallel Clustal W/HT Clustal.

## Acknowledgments

Special thanks to other members of the SGI ChemBio team Juli Nash Moultray, David Zirl, Joe Landman, Dan Stevens, and Stavros Taraviras. Thanks to Julie Thompson-Maaloum (IGBMC) and Toby Gibson (EMBL) for very useful help with the original Clustal W, to Jeff Yuan and Rick Blevins of Merck for sharing the MULTICLUSTAL program, and to Rodrigo Lopez of European Bioinformatics Institute for feedback and testing Parallel Clustal W on the EBI SGI Origin series server.

## References

1. Thompson, J.D., Higgins, D.G., and Gibson T.J. CLUSTAL W: improving the sensitivity of progressive multiple alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.* 22, 4673, 1994.
2. Yuan, J., Amend, A., Borkowski, J., DeMarco, R., Bauley, W., Liu, Y., Xie, G., and Blevins, R. MULTICLUSTAL: a systematic method for surveying Clustal W alignment parameters. *Bioinformatics*, 15(10), 862, 1999.
3. Saitou, N., and Nei, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol. Evol.*, 4, 406, 1987.
4. [www.openmp.org](http://www.openmp.org)
5. Amdahl, G. Validity of the single-processor approach to achieving large-scale computing capabilities. *Proc. AFIPS Conf.*, 1967.
6. SGI 2000 Bioinformatics Performance Report can be downloaded from [www.sgi.com/solutions/sciences/chembio](http://www.sgi.com/solutions/sciences/chembio)

© 2001 Silicon Graphics, Inc., All rights reserved. Specifications subject to change without notice. Silicon Graphics is a registered trademark and SGI, OpenMP, and Origin and the SGI logo are trademarks of Silicon Graphics, Inc. Linux is a registered trademark of Linus Torvalds. All other trademarks mentioned herein are the property of their respective owners.