# Enhancing Parallelism of Pairwise Statistical Significance Estimation for Local Sequence Alignment

Yuhong Zhang[1,2], Md. Mostofa Ali Patwary[2], Sanchit Misra[2],
Ankit Agrawal[2], Wei-keng Liao[2], and Alok Choudhary[2]

[1]School of Computer Science and Engineering,
University of Electronic Science and Technology of China, Chengdu, China
Email: yuhongzhang@uestc.edu.cn
[2]Department of Electrical Engineering and Computer Science, Northwestern University, USA
Email: {mpatwary, smi539, ankitag, wkliao, choudhar}@eecs.northwestern.edu

***Abstract*—Pairwise statistical significance (PSS) has been found to be able to accurately identify related sequences (homology detection), which is a fundamental step in numerous applications relating to sequence analysis. Although more accurate than database statistical significance, it is both computationally intensive and data intensive to construct the empirical score distribution during the estimation of PSS, which poses a big challenge in terms of performance and scalability. Multicore computers and clusters have become increasingly ubiquitous and more powerful than before. In this paper, we evaluate the use of OpenMP, MPI and hybrid paradigms to accelerate the estimation of PSS of local sequence alignment. Through distributing the compute-intensive kernels of the pairwise statistical significance estimation procedure across multiple computational units, we achieve a speedup of up to $113.10\times$ using $128$ cores.**

***Keywords*-Pairwise statistical significance; Multicore, OpenMP; MPI; Hybrid;**

## I. Introduction

The recent decades have witnessed dramatic increase in the quantity and variety of publicly available proteomic and genomic sequence data. GenBank, for example, as of August $2011$, has accumulated more than $10^{12}$ nucleotides of nucleic acid sequence data, and continues to grow at an exponential rate, approximately doubling every $18$ months [1]. How to deal with the massive quantities of data pouring from the sequencing factories, make sense of them, and render them accessible to people who are working on a wide variety of problems is a big challenge in bioinformatics [2, 3].

Pairwise sequence alignment (PSA) is widely used in the analysis of DNA and protein sequences [4, 5]. It builds the basic platform for many other biological applications such as homology detection, protein structure prediction, finding protein function and deciphering evolutionary relationships. Sequence alignment is an effective method that reports a score, indicating the relatedness between sequences. Generally, a higher score indicates that the sequences are more related. However, the alignment score depends on various factors like the alignment program, scoring scheme, sequence lengths, and compositions of sequence under comparison [6]. It may not make sense to draw a conclusion about the relatedness of pairwise sequences from scores alone. Therefore, it is more appropriate to measure the quality of a PSA using the statistical significance of the score rather than the score itself [7]. An alignment score is more statistically significant if it has a low probability of occurring by chance. Statistical significance of sequence alignment scores is very important to know whether an observed sequence similarity could imply a functional or evolutionary link, or is a chance event [6]. Pairwise statistical significance (PSS) is a promising method to evaluate the statistical significance of an alignment, which is specific to the sequence-pair being aligned, and independent of database [8]. However, the estimation of PSS is very data-intensive and computation-intensive [9]. Therefore, applying high performance computing (HPC) techniques is an obvious choice to accelerate the estimation.

Although FPGAs [10] and GPUs [11] have been used to accelerate the estimation of pairwise statistical significance, the researchers in bioinformatics community are required to acquire special knowledge to use them properly. Moreover, special hardware requirement limits the utilization of those high performance platforms. On the other hand, the multicore computers or laptops and clusters have become increasingly ubiquitous and more powerful. Therefore, it is of interest to use high performance technologies to unlock the potential of computers or laptops and clusters. Based on this observation and motivation, in this paper, we present OpenMP, MPI and hybrid (OpenMP + MPI) implementations to accelerate the estimation of PSS. After careful performance analysis, we have efficiently distributed the compute-intensive kernels of the algorithm across processors (cores), so as to reap the maximum benefits of OpenMP or/and MPI paradigms. Our experiments show that our parallelization methodology achieves high performance for these applications. The maximum speedup of OpenMP, MPI and hybrid implementations are $18.94\times$, $22.58\times$ and $22.65\times$,

respectively, when using single node with 24 cores. With 128 cores spread across 6 nodes, pure MPI implementation and hybrid implementation achieve speedups of $113.10\times$ and $112.86\times$ respectively. The results of pairwise statistical significance estimation depend on the sequence alignment technique used.

The remainder of this paper is organized as follows. We present the essential background about PSS and parallel programming paradigm in Section II. Subsequently, we describe the different parallel implementations in Section III. We present the experimental results and performance analysis of different implementations in Section IV, followed by conclusions in Section V.

## II. Background

### A. Algorithm overview

In order to determine whether the resulting similarity (usually representing as a alignment score) between the two sequences implies an evolutionary link or is just a chance event, one has to know how probable it is to obtain the same score by aligning completely unrelated (e.g., randomly chosen) sequences [12]. Thus, statistical approaches are often used to avoid a bias brought by single sample. These approaches are based upon theoretical models or permutation reconstructions of the observed sequences and convert the scores into a *P-value* or *E-value* [13]. Accurate estimation of statistical significance of sequence alignment has attracted a lot of research in recent years [8, 14–19].

In the case of gapless alignment, an asymptotic theory for local alignment scores has been developed rigorously. The optimal alignment scores $S$ among random, unrelated sequences approximately turns out to be a Gumbel or an Extreme Value Distribution (EVD) [6, 12, 13, 20], which has a much broader tail than that of the Gaussian distribution. The *P-value*, defined as the probability of observing a gapless alignment of a sequence pair with a score $S$ greater than $x$, is simply given by

$$P(S > x) \approx 1 - exp(-Kmne^{-\lambda x}) = 1 - e^{-E(x)} \quad (1)$$

where $m$ and $n$ are the lengths of the two sequences being aligned, $\lambda$ and $K$ are estimable constants depending on the scoring scheme and average compositions of sequences based on the Poisson distribution hypothesis [21], and $E(x)$, also known as *E-value*, is the expected number of distinct local alignments with score values of at least $x$.

For the case of gapped alignment, although no asymptotic score distribution has yet been established analytically for the local alignments, simulations and computational experiments strongly indicate that, for both local and global alignments, these scores still roughly follow the Gumbel law after pragmatic estimation of the $\lambda$ and $K$ parameters [6, 22–24].

Pairwise statistical significance is an attempt which makes the estimation process more specific to the sequence pair being compared [9]. Considering one sequence pair $q$ (known as query sequence) and $s$ (known as subject sequence), their lengths $m$ and $n$, respectively, given the scoring scheme $SC$ (substitution matrix, gap opening penalty, gap extension penalty) and the number of permutations $N$, the PSS described in [8] can be thought of being estimable by the following function:

$$PSS(q, s, m, n, SC, N)$$

The function PSS simulates $N$ random sequences of equal length n through permuting $s$ $N$ times, generates $N$ scores by aligning $q$ against the $N$ the permutated copies of $s$, then fits these scores to an extreme value distribution using type-I censored maximum likelihood fitting [25] to obtain the statistical parameters $K$ and $\lambda$, and finally returns the estimation of statistical significance of the pairwise alignment score between $q$ and $s$.

Although the estimation of PSS has been shown to be accurate [9], it involves thousands of such permutations and alignments, which are enormously time consuming and can be impractical for estimating PSS for a large number of sequence pairs. For instance, for our previous experiments with 86 query sequences and 2771 subject sequences, the sequential implementation takes more than 32 hours [26]. Hence, using multicore computers or clusters is highly conducive to accelerating the computation of PSS. Moreover, large data sets demand more computing power. The earlier work [9] only presents an MPI implementation for computing single pair estimation of PSS. But in this work, we not only extend that work to support multi-pair estimation of PSS using different strategies, but also design a pure OpenMP and OpenMP/MPI hybrid implementation to enhance the parallelism, maximizing the benefits of multicore era, as also illustrated by the results.

### B. Parallel Programming paradigm

In recent years, the several CPU vendors, such as AMD® and Intel®, have shifted gears away from heading for more clock speeds to adding parallelism support on-chip with multicore processors (i.e., putting more simpler and smaller cores on a single chip compared to before). This practice can bypass many of the technological obstacles (such as energy and other constraints on processor designs) that CPU vendors are encountering while trying to boost speeds. But only if an application takes advantage of these multiple cores, it is able to harvest the benefits. This is where OpenMP paradigm sets foot in this picture.

Currently, the most often-used parallel programming paradigm is the *flat MPI* mode, in which each single-threaded MPI process is executed on one core. However, the growth in memory capacity is not keeping pace with the

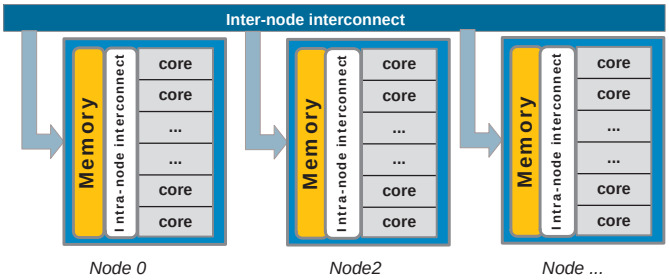


Figure 1: The hybrid parallel programming paradigm

growth in the number of cores. Therefore, this programming model of MPI everywhere is not likely to be a viable model on these newer architectures simply because of the reduced amount of memory per core [27]. It is important to investigate a hybrid paradigm (for example, a combination of OpenMP and MPI) to avoid these issues [28].

The hybrid paradigm can match the characteristics of the cluster of multicore nodes very well, since it allows for a two-level communication pattern (i.e., intra-communications among cores within a node and inter-node communications among different nodes), as shown in Figure 1. The main benefits of the hybrid paradigm is to take advantage of process level coarse-grain parallelism, in which one MPI process executes on each multi-core processor, and fine grain parallelism on a loop level, in which each MPI process spawns a team of threads to occupy the multicore processor when encountering parallel sections of code using OpenMP.

## III. Implementations

For a given single sequence pair ($q$, $s$), careful analysis of the data pipelines of PSS estimation shows that its computation can be decomposed into three computation kernels: (i) Permutation: generating $N$ random sequences by permuting $s$; (ii) Alignment: aligning the $N$ permuted sequences of $s$ against $q$ using Smith-Waterman (SW) algorithm [22]; and (iii) Fitting: obtaining statistical constants $K$ and $\lambda$ by fitting the $N$ scores generated in (ii) into an EVD, finally returning the PSS between sequence pair $q$ and $s$ according to Equation 1. The kernels of permutation and alignment comprise the overwhelming majority of the overall execution time (varying from $92.4\%$ to $99.2\%$). Therefore, efforts should be spent to optimize these two kernels, which will result in net significant improvements to the performance of the algorithm as a whole.

### A. OpenMP implementation

Algorithm 1 outlines the basic idea of parallelizing the estimation of PSS using OpenMP. For each query $q$, we load the standard substitution matrix (SSM) like BLOSUM62 or position-specific scoring matrix (PSSM) (line 3). Subsequently, for each subject sequence $s$, we need to run a loop of $N$ iterations (line 5-8). This loop is parallelized using

**Algorithm 1** Pseudo-code of OpenMP implementation for PSS estimation.
**input:** $Q$: query sequence database; $S$: subject sequence database;
**output:** $PSS$: Pairwise statistical significance

```
1:  Read sequences from database (Q, S)
2:  for each query sequence q ∈ Q do                          ▷ A
3:      Read substitution matrix M for the given query q;
4:      for each subject sequence s ∈ D do                    ▷ B
5:          for k ← 0, N; in parallel do                      ▷ C
6:              s_k ← k-th permuted sequence of s
7:              Scores [k] ← SW (q, s_k, m, n, SC)
8:          end for
9:          (K, λ) ← EVD_Fitting (Scores)
10:         PSS (i) ← 1 − exp(−Kmne^{−λx})
11:     end for
12: end for
```

OpenMP. In each iteration of the loop, we compute a new permutation of the subject sequence $s$, called $s_k$ (line 6), and use SW algorithm to align the permuted sequence, $s_k$, to the query sequence, $q$, to get a score (line 7).

Then the $N$ alignment scores are gathered together on the master thread and fitted into an EVD (line 9). Finally, the PSS is obtained using equation 1 (line 10). In this case, each thread works on the alignment of one pair of sequences. Multiple alignment tasks are performed in parallel by different threads. As shown in Algorithm 1, there are three *for-loop* levels (marked by 'A', 'B', 'C', respectively) that can be parallelized. The computation of SW algorithm is very sensitive to the length of query and subject sequences, it will result in an imbalanced workload among CPU cores if 'A' or 'B' *for-loop* is parallelized, which would hurt the performance. By comparison, in 'C' *for-loop*, a particular query is aligned to the permuted copies of a particular subject sequence. Hence the lengths of the two sequences do not vary between the iterations of the for loop, which means *parallelizing* this level *for* will obtain a good load balance among cores. More fine-grained parallelization is also possible, where multiple threads collaborate to align one pair of sequences (i.e., in intra-parallelism mode [11]). However, because of the overhead of synchronization and communications, it is difficult to achieve good performance. Our experiments with intra-parallelism show no further performance improvement as expected.

### B. MPI implementation

*1) Intuitive strategy:* As discussed in the previous section, the kernels of permutation and alignment are independent of each other during the pairwise statistical significance estimation procedure, which map very well to programming models capable of expressing MPI task parallelism. Let us assume that the number of nodes is $P$ and the number of permutations is $N$. In [9], the *root* process broadcasts a query ($q$) and a subject sequence ($s$) to all other processes

in each iteration. Then, each process computes $\lceil N/P \rceil$ permutations of $s$ and aligns them to the query sequence, $q$. Subsequently, the *root* process gathers the $\lceil N/P \rceil$ alignment scores from each process and uses them for fitting. The intuitive strategy for multi-pair estimation of PSS is simply extended from [9]. That paper discusses the parallelization of estimation of PSS of a single pair of query and subject sequences. We just execute that for each query and subject sequence pair. The main disadvantage of this policy is that a higher overhead of communication among processes has to be paid, as the query and subject sequences have to broadcasted to all the processes. Also, communication is required for gathering alignment scores for each pair. Moreover, only the *root* performs the task of fitting.

*2) Tiling strategy:* Instead of distributing the permutation and alignment tasks for each pair of query and subject sequences like in the intuitive strategy, the tiling strategy employs a more coarse-grained parallelism. Tiling strategy partitions the sequence database into different disjoint sets of subject sequences and assigns one set to each process. Therefore, each process has a subset of subject sequences and all the query sequences and can estimate the PSS of each pair independent of other processes. This enhances data locality on single node and reduces the overhead of communication.

In order to achieve higher efficiency for this parallelization, the workload of each node should be roughly equal. The workload of the estimation of PSS process mainly depends on the length of query and subject sequence. Before tiling the sequences, we therefore reorder the subject sequences based on their lengths.

Moreover, instead of distributing roughly equal number of subject sequences to different processes, this method partitions sequences based on the total length of sequences. This is done as follows. Before all the sequences are partitioned, they are sorted by their length. Let $bin[i]$ represent the set of subject sequences assigned to process $i$. We populate the bins in the following way: (i) In the first round, we assign the first sequence from the sorted list to $bin[0]$, and the second sequence to $bin[1]$, and so on until we assign the $p$-th sequence to $bin[p]$, where $p$ is the total number of nodes. (ii) In each following round, we get the sum of the lengths of all the sequences in each bin, say $L[i]$. We assign the smallest length sequence to the bin with maximum $L[i]$, assign the second smallest one to the bin of second maximum $L[i]$, and so on. (iii) Continue the step (ii) until all database sequences are allocated to the bins. After that, each node receives the nearly same amount of sequences. This helps to balance the workload across nodes and improve the performance. Unlike the previous strategy, there is no need to gather alignment scores generated from every node as each node can estimate the pairwise statistical significance separately. The overhead of communication is much smaller, which is expected to result in a higher performance and better scalability.

**Algorithm 2** Pseudo-code of Hybrid Implementation for PSS estimation.
**input:**$Q$: query sequence database; $S$: subject sequence database;
**output:** $PSS$: Pairwise statistical significance

```
1: read sequences from database (Q, S)
2: if Rank = 0 then
3:     Sort subject sequences based on their length
4:     Partition subject sequences into roughly equal bins based
   on the aggregate of their lengths.
5:     Transfer each bin to the corresponding MPI task.
6: end if
7: for each query sequence q ∈ Q do
8:     Read substitution matrix M for the given query q
9:     for each subject sequence s ∈ bin[i]; in parallel do
                                        ▷ MPI-level parallization
10:        for k ← 0, N; in parallel do
                                        ▷ OpenMP-level parallization
11:            s_k ← k-th permuted sequence of s
12:            Scores [k] ← SW (q, s_k, m, n, SC)
13:        end for
14:        (K, λ) ← EVD_Fitting (Scores)
15:        PSS (i) ← 1 − exp (−Kmne^{−λx})
16:    end for
17: end for
```

### C. Hybrid implementation of OpenMP and MPI

In essence, hybrid implementation is a combination of our OpenMP and MPI (tiling strategy) implementation. Algorithm 2 provides a pseudo-code of the hybrid implementation. The hybrid implementation consists of a heirarchy of MPI tasks and OpenMP threads. We use the tiling strategy to distribute workload over multiple MPI tasks. Therefore, an MPI task $i$ handles the partition $bin[i]$ of the subject sequences (line 9). When MPI task $i$ picks up one pair of sequences $(q, s)$ belonging to itself, it spawns several threads, which cooperate to complete the kernels of permutation and alignment in parallel (line 10). After all the alignments are done, the threads exit and MPI task $i$ uses the alignment scores to perform fitting and compute PSS (line 14-15). Note that, thread-level synchronization (explicit and implicit) and communications bring extra overhead in hybrid paradigm, which means hybrid paradigm do not necessarily make a higher performance improvement than flat-MPI paradigm. However, OpenMP paradigm can dramatically decrease memory usage, allowing larger problems to be addressed. Therefore, hybrid mode can find a tradeoff between speedup and memory consumption.

## IV. Experimental results and discussion

### A. Experimental setup

We carried out the experiments on Cray XE6 machines (provided by Hopper system at NERSC), each with 24 cores and 32 GB memory. Each node contains two twelve-core AMD® "Magny-Cours" @2.1 GHz processors. The

operating system is 64-bit Linux. The sequences data used in this work comprise of a non-redundant subset of the CATH 2.3 database [29, 30]. This dataset consists of 2771 domain sequences as our subject sequences library, which represents 1099 homologous superfamilies and 623 topologies and includes 86 CATH queries serving as our query set.

### B. Results and discussion

In this subsection, we present the performance differences among different programming paradigms on the same platform. Furthermore, we analyze potential reasons for these performance differences. The performance on one node using OpenMP, MPI and their hybrid paradigm are investigated first. Then we present results for performance on multiple nodes.

In order to understand the performance characteristics of the implementations, we use a simple performance model $t_{total} = t_{parallel} + t_{serial} + t_{overhead}$, where $t_{total}$ is the total running time, $t_{parallel}$ is the running time of the section that can be parallelized, $t_{serial}$ is the running time of section that runs sequentially and $t_{overhead}$ is the extra overhead (such as communication or synchronization) when using OpenMP and/or MPI paradigm.

*1) Performance analysis on one node:* In this subsection, we compare the performance of these three paradigms and analyze the reasons of differences of performance. All of the experiments are executed on single node with 24 cores. The amount of computation required for a query sequence depends on its length. Hence we compared the performance using multiple lengths of query sequences. Four query sequences of length 200, 400, 800, and 1600 are chosen from CATH to align against all the 2771 subject sequences. We have tested different number of OpenMP threads, $T$, and MPI tasks, $P$. Figure 2 shows the experimental results of the OpenMP implementation. All speedups are computed over the corresponding sequential implementation. The maximum speedup is $6.10\times$, $9.61\times$, $14.55\times$, and $18.94\times$, corresponding to the length of query equal to 200, 400, 800 and 1600, respectively. Observe that, performance increases with the increase in length of query sequences.

We use CrayPat, a performance analysis tool for the XT and XE platforms, to profile the sequential implementation. Let $f = t_{parallel}/t_{total}$ present the percentage of whole implementation that can be parallelized, in our case, CrayPat shows that $f$ (i.e., the percentage of computation kernels of permutation and alignment) varies from $92.4\%$ to $99.2\%$ corresponding to the query sequence lengths changing from 200 to 1600. Therefore, according to Amdahl's law, the maximum speed of the OpenMP implementation in theory falls between $8.73$ to $20.27\times$ when number of processors (cores) $P$ is 24. In practice, the extra overhead brought by using parallel paradigm will further lower performance. This explains that it is hard to achieve a high speedup for an

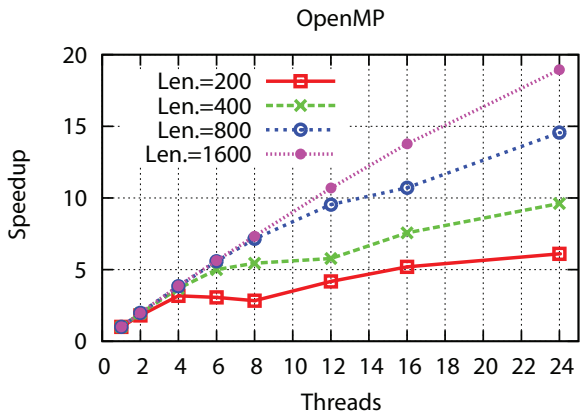


Figure 2: The speedup of the OpenMP implementation

OpenMP implementation unless $f$ becomes the dominant part.

Additionally, although in most cases, the non-uniform memory architecture (NUMA) used by Hopper machine provides advantages over traditional symmetric multiProcessing (SMP) solutions in terms of low level memory characteristics (such as memory bandwidth) and scalability, it also brings its disadvantages, like the NUMA effect [28]. Each NUMA node we are using is essentially a four-chip node, one of which contains a six-core die. All the memory in NUMA is transparently accessible, but if a process running on chip node $i$ accesses the memory connected to a different chip node $j$ (where $i \neq j$), a performance penalty will be incurred (i.e., NUMA effect). When $f$ is lower, the performance will be more sensitive to this extra overhead. This can shed a light on why the speedup using 8 threads is even a little lower than using 6 threads when the length of query sequence is equal to 200, because the benefits of increasing threads is lower than the overhead of NUMA effect. But when the length of query sequence increases, the NUMA effect decreases. In [28], authors claimed that it is very hard to address the performance implications of the NUMA effect while using more than 6 OpenMP threads on Hopper. Our experiments confirmed this claim again.

As for MPI implementation using tiling strategy, due to the requirement of very limited communications among processes and zero dependency of the three kernels (permutation, alignment and fitting) of different query-subject sequence pairs, it achieves an almost linear speedup and similar scalability, resulting in curves for the four different query lengths superimposing each other, as shown in Figure 3. Therefore, the performance of the MPI implementation is not sensitive to the length of query sequence. When $P = 24$, the speedups are obtained as $22.50\times$, $22.58\times$, $22.57\times$, and $22.58\times$, corresponding to the length of query sequence $200, 400, 800$, and $1600$, respectively.

For the experiments of hybrid implementation, we only test different combinations of threads and MPI tasks, subject

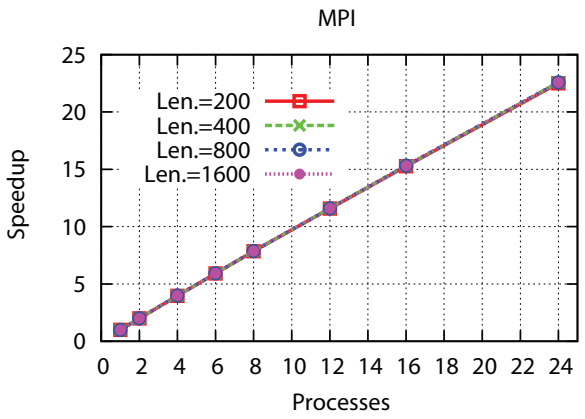


Figure 3: The speedup of the MPI implementation (using the tiling strategy) on single node

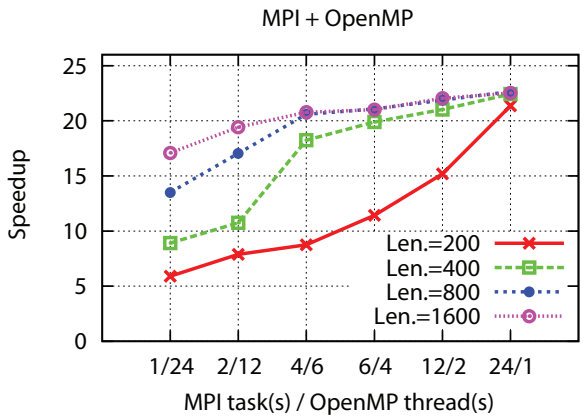


Figure 4: The speedup of hybrid implementation on single node using different combination of MPI task(s) and OpenMP thread(s)

to $T \times P = 24$. When $T = 24$ and $P = 1$, the performance of hybrid implementation is roughly equal to the pure OpenMP one. On the other hand, when $T = 1$ and $P = 24$, its performance is nearly same as flat MPI one, with a maximum speedup of $22.51\times$. In other words, the performance of hybrid implementation is between pure OpenMP and MPI. When the length of query sequence $L$ is 200, representing lower computation, its performance is more sensitive to the $T$. This is because more OpenMP threads will bring more scheduling overhead, which leads to an increase in the fraction $(1 - f)$, consequently a lower speedup according to Amdahl's law. But when $L > 400$ and $T <= 6$, as one decreases $P$ and increases $T$, the overall memory usage decreases (a more than 50% reported by CrayPat) significantly but performance loss is very limited (less than 10%), as shown in Figure 4. Another observation in this figure is that using more than 6 OpenMP threads incurs quite a substantial performance penalty because of NUMA effects.

In short, using more OpenMP threads can potentially save memory usage while it may also increase the synchronization cost (implicit or explicit) among these threads and the activation/deactivation overhead. Therefore, it is hard to generalize a common rule for setting of the optimized number of OpenMP threads to deliver the best performance, because it also depends on specific computer architectures, platforms and problem size.

*2) Performance analysis on multiple nodes:* In this case, we run the entire query set containing 86 queries against the entire CATH 2.3 database containing 2771 subject sequences. Currently, OpenMP paradigm cannot be applied on more than one nodes on Cray platforms. Therefore, in this subsection, we only compare the performance of flat-MPI and hybrid paradigm. Due to limited communications and lower dependency across MPI tasks compared to the intuitive strategy, and also due to better load balancing, the flat-MPI implementation using the tiling strategy results in

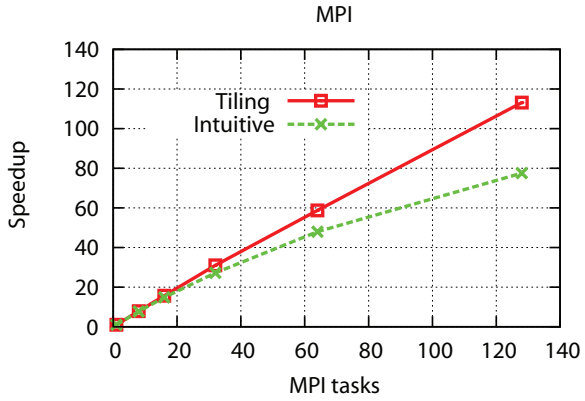


Figure 5: The speedup of MPI implementation on multiple nodes using tiling and intuitive strategy

a better scalability and higher performance. Their maximum speedups are $113.09\times$ and $77.54\times$, respectively when using 128 MPI tasks, as shown in Figure 5.

As for hybrid implementation on multiple nodes, the experiments are carried for different combinations of MPI tasks $P$ and OpenMP threads $T$, subject to $P \times T = 128$. Similar to the experiments of single node, when $T = 1$ and $P = 128$, its performance is up to a maximum speedup of $112.86\times$. Again, substantial reductions in memory usage occurs with increasing number of OpenMP threads at the cost of some performance loss. When $T > 6$, NUMA effect hurts performance significantly for the same reason analyzed in above subsection. We show the results in Figure 6.

In summary, MPI paradigm usually obtains best performance and scalability but consumes more memory. Hybrid paradigm may not alter the performance of an application much, but it can decrease memory usage, in some cases, very dramatically. It allows larger problems to have a better chance to be addressed, which is especially important for

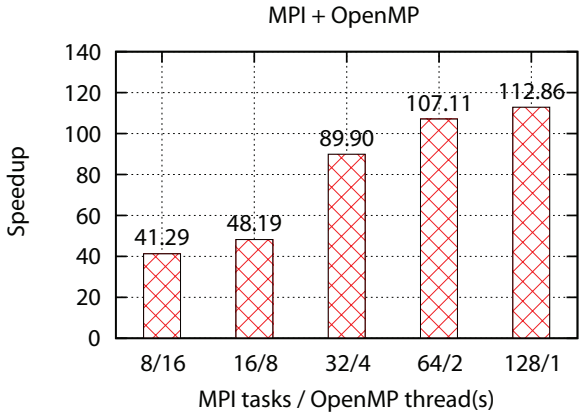


Figure 6: The speedup of hybrid implementation of multiple nodes using different combinations of MPI tasks and thread(s)

the current trend in computer architecture development.

All the implementations of the proposed method and related programs are available for free academic use at http://cucis.ece.northwestern.edu/projects/PSSE/.

## V. Conclusions

In this paper, we evaluate the use of OpenMP, MPI and hybrid paradigms to build a high performance accelerator to estimate the pairwise statistical significance of local sequence alignment. By distributing the most compute-intensive task of permutation and alignment, and in some cases fitting, our accelerator reaps the benefits of these parallelization paradigms, which results in high end-to-end speedups for estimation of PSS. The proposed efficient framework is also applicable to a wide variety of next-generation sequencing comparison based applications, such as DNA sequence mapping and database search. As the size of biological sequence databases are increasing rapidly, even more powerful high performance computing accelerator platforms, comprising of heterogeneous components such as multi-core CPU along with many-core GPU clusters and possibly FPGAs, are expected to be more and more common and imperative for sequence analysis, for which this work could be a significant stepping stone.

## Acknowledgment

This work is supported in part by NSF award numbers CCF-0621443, OCI-0724599, CCF-0833131, CNS-0830927, IIS-0905205, OCI-0956311, CCF-0938000, CCF-1043085, CCF-1029166, and OCI-1144061, and in part by DOE grants DE-FC02-07ER25808, DE-FG02-08ER25848, DE-SC0001283, DE-SC0005309, DE-SC0005340, National Nature Science Foundation of China (Contract No. 60973118) and China Scholarship Council. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

## References

[1] NCBI, "Genbank release notes," 2011, availabe at:ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt.

[2] D. S. Roos, "COMPUTATIONAL BIOLOGY: Bioinformatics–Trying to Swim in a Sea of Data," *Science*, vol. 291, no. 5507, pp. 1260–1261, 2001.

[3] P. M and J. CV, "Making sense of score statistics for sequence alignments," *Briefings in Bioinformatics*, vol. 2, no. 1, pp. 51–67, 2001.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Local Alignment Search Tool." *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[5] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs." *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[6] R. Mott, "Accurate Formula for P-values of Gapped Local Sequence and Profile Alignments." *Journal of Molecular Biology*, vol. 300, pp. 649–659, 2000.

[7] M. AY and B. M, "Statistical significance in biological sequence analysis," *Briefings in Bioinformatics*, vol. 7, no. 1, pp. 2–24, 2006.

[8] A. Agrawal, V. Brendel, and X. Huang, "Pairwise statistical significance versus database statistical significance for local alignment of protein sequences," in *Bioinformatics Research and Applications*, ser. LNCS(LNBI), vol. 4983. Springer Berlin/Heidelberg, 2008, pp. 50–61.

[9] A. Agrawal, S. Misra, D. Honbo, and A. N. Choudhary, "Mpipairwisestatsig: parallel pairwise statistical significance estimation of local sequence alignment," in *HPDC*, 2010, pp. 470–476.

[10] D. Honbo, A. Agrawal, and A. N. Choudhary, "Efficient Pairwise Statistical Significance Estimation using FPGAs," in *BIOCOMP*, 2010, pp. 571–577.

[11] Y. Zhang, S. Misra, D. Honbo, A. Agrawal, W. keng Liao, and A. N. Choudhary, "Efficient pairwise statistical significance estimation for local sequence alignment using GPU," in *ICCABS*, 2011, pp. 226–231.

[12] R. Bundschuh, "Rapid significance estimation in local sequence alignment with gaps," *Journal of Computational Biology*, vol. 9, no. 2, pp. 243–260, 2002.

[13] S. Karlin and S. F. Altschul, "Methods for Assessing the Statistical Significance of Molecular Sequence Features by Using General Scoring Schemes," *PNAS, USA*, vol. 87, no. 6, pp. 2264–2268, 1990.

[14] A. Agrawal and X. Huang, "Pairwise statistical significance of local sequence alignment using sequence-specific and position-specific substitution matrices,"

*IEEE/ACM Trans. Comput. Biology Bioinform.*, vol. 8, no. 1, pp. 194–205, 2011.

[15] ——, "Pairwise Statistical Significance of Local Sequence Alignment Using Multiple Parameter Sets and Empirical Justification of Parameter Set Change Penalty," *BMCB*, vol. 10, no. Suppl 3, p. S1, 2009.

[16] ——, "PSIBLAST_PairwiseStatSig: reordering PSI-BLAST hits using pairwise statistical significance," *Bioinformatics*, vol. 25, no. 8, pp. 1082–1083, 2009.

[17] A. Poleksic, J. F. Danzer, K. Hambly, and D. A. Debe, "Convergent Island Statistics: a fast method for determining local alignment score significance," *Bioinformatics*, vol. 21, no. 12, pp. 2827–2831, 2005.

[18] S. Sheetlin, Y. Park, and J. L. Spouge, "The gumbel pre-factor k for gapped local alignment can be estimated from simulations of global alignment," *Nucleic Acids Research*, vol. 33, no. 15, pp. 4987–4994, 2005.

[19] A. Agrawal, A. Choudhary, and X. Huang, "Sequence-specific sequence comparison using pairwise statistical significance," in *Software Tools and Algorithms for Biological Systems*, ser. Advances in Experimental Medicine and Biology, H. R. R. Arabnia and Q.-N. Tran, Eds. Springer New York, 2011, vol. 696, pp. 297–306.

[20] O. Bastien and E. Marechal, "Evolution of biological sequences implies an extreme value distribution of type I for both global and local pairwise alignment scores," *BMCB*, vol. 9, no. 1, p. 332, 2008.

[21] A. Dembo, S. Karlin, and O. Zeitouni, "Limit distribution of maximal non-aligned two-sequence segmental score," *Ann. Prob*, vol. 22, pp. 2022–2039, 1994.

[22] M. S. Waterman and M. Vingron, "Rapid and Accurate Estimates of Statistical Significance for Sequence Database Searches," *PNAS, USA*, vol. 91, no. 11, pp. 4625–4628, 1994.

[23] W. R. Pearson, "Empirical Statistical Estimates for Sequence Similarity Searches," *J. of Molecular Biology*, vol. 276, pp. 71–84, 1998.

[24] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences," *J. of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[25] S. R. Eddy, "Maximum likelihood fitting of extreme value distributions," 1997, unpublished work.

[26] Y. Zhang, S. Misra, A. Agrawal, M. M. A. Patwary, W. keng Liao, and A. N. Choudhary, "Accelerating pairwise statistical significance estimation for local alignment by harvesting GPU's power," *BMC bioinformatics*, 2011, accetped.

[27] H. Shan, H. Jin, K. Fuerlinger, A. Koniges, and N. J. Wright, "Analyzing the Effect of Different Programming Models upon Performance and Memory Usage on Cray XT5 Platforms," in *Cray User's Group Meeting 2010*, 2010.

[28] N. J. Wright, H. Shan, F. Blagojevic, H. Wasserman, T. Drummond, J. Shalf, K. Fuerlinger, K. Yelick, S. Ethier, M. Wagner, N. Wichman, S. Anderson, and M. Aamodt, "The NERSC-Cray center of excellence: Performance optimization for the multicore era," in *Cray User's Group Meeting 2011*, 2011.

[29] C. Orengo, A. Michie, S. Jones, D. Jones, M. Swindells, and J. Thornton, "CATH - a hierarchic classification of protein domain structures," *Structure*, vol. 5, no. 8, pp. 1093 – 1109, 1997.

[30] M. L. Sierk and W. R. Perrson, "Sensitivity and selectivity in protein structure comparison," *Protein Science*, vol. 13, no. 3, pp. 773–785, 2004.