2017 IEEE International Parallel and Distributed Processing Symposium Workshops

Par-eXpress: A Tool for Analysis of Sequencing Experiments With Ambiguous Assignment of Fragments in Parallel

Mucahid Kutlu
Dept. of Computer Science & Eng.
Qatar University
Doha, Qatar
Email: mucahidkutlu@qu.edu.qa

Gagan Agrawal
Dept. of Computer Science & Eng.
Ohio State University
Columbus, OH, 43210
Email: agrawal@cse.ohio-state.edu

James S. Blachly
Comprehensive Cancer Center
Ohio State University
Columbus, OH, 43210
Email: james.blachly@osumc.edu

***Abstract*—With new high-throughput and low-cost sequencing technologies, an increasing amount of genetic data is becoming available to researchers. While the analysis of this vast amount of data has great potential for future scientific advances, it becomes imperative to exploit parallelism in order to process this data efficiently.**

In this paper, we address probabilistic assignment of ambiguously mapped fragments. This is a very significant, but time consuming, process for downstream analysis of genomic data. We develop a distributed-memory parallel version of a popular probabilistic fragment assignment tool, eXpress. In our experiments, we show that our approach achieves significant speedups over eXpress without decreasing its accuracy. The speedup we achieve increases as the number of iterations and/or data size increases.

I. Introduction

With recent developments in sequencing technologies, genomic data can be obtained at a much faster rate and at a lower cost. The researchers are able to access vast amount of genomic data through various projects. A noteworthy example of such efforts is the 1000 Genomes Project[1], which run between 2008-2015 and collected genomic data of 2,504 individuals over 26 populations. The project produced more than 300TB of data, which brings huge computational challenges in its further analysis.

A typical modern sequencing experiment, involves four main steps: 1) Dividing genetic materials (*e.g.*, DNA or cDNAs) into fragments, 2) Processing with sequencing machines to transfer the data into digital world, 3) Reconstructing the genome by aligning fragments or mapping them to a reference genome/sequences. 4) Downstream analysis of the resultant data. Due to limited amount of unique parts in reference sequences, exact locations of some fragments may not be identified, causing ambiguously assigned fragments. This ambiguity makes downstream analysis of the data harder in RNA-Seq, ChiP-seq and metagenomics experiments.

[1]www.1000genomes.org

A popular approach to solve this problem is probabilistic assignment of fragments to their potential target sequences by using the *Expectation-Maximization* (EM) algorithm [12], [5], [14], [11]. *eXpress* [12] is one of the most popular EM-based tools for ambiguous fragment assignment. eXpress employs a shared memory parallelization in which a thread is dedicated to load the fragments into memory, another thread is dedicated to learn *auxiliary parameters* and other threads are assigned for applying the EM algorithm. eXpress has several advantages over previous approaches [12]. First, it is more efficient than RSEM [5] and Cufflinks [14]. Second, it achieves a constant memory consumption while RSEM's and Cufflinks's memory requirements increase as the number of fragments increases. In terms of accuracy, eXpress outperforms others with small number of fragments ($\leq$20M as reported in [12]), and is more robust than RSEM and Cufflinks at low depth. RSEM slightly outperforms eXpress when number of fragments is more than 20M.

Even eXpress has several advantages, it is still time consuming to process large-scale genomic data, due to its limited parallelization. The available eXpress implementation employs only shared memory parallelization and maximum 3 threads can be used effectively. In addition, in order to keep memory consumption constant, it reads the fragments from the disk at each iteration, which can be very costly as the number of iterations increases. Therefore, a distributed-memory parallel version of eXpress algorithm is required to process large-scale genomic data efficiently.

A noteworthy study about distributed-memory parallelization of eXpress algorithm is eXpress-D [11], which employs Spark [16] to solve the problem of re-loading the data at each iteration. However, eXpress-D cannot process genomic data that contains insertions or deletions (i.e. indels). Due to this major drawback of eXpress-D, it cannot be used in genomic research areas that deal with indels (e.g. cancer research).

In this work, we develop *Par-eXpress*, which is a distributed-memory parallel version of eXpress software. Par-eXpress does not put any restriction on input data. In order to preserve the accuracy, we use the original shared-memory parallelization of eXpress in the first iteration, which has a different processing structure than the other iterations. Actual distributed-memory parallelization begins after the first iteration. We distribute the data among the processes such that only a small portion of the processes need to synchronize at each iteration and share their updated parameters. The rest of the processes can continue their execution independently. Each process works on the same assigned data chunk in each iteration. Therefore, processes can keep their assigned data chunks in their memories, and expensive memory loading is avoided. Once all processes finish their tasks, the partial results are combined to form the final output.

In our experiments, we show that Par-eXpress is able to achieve significant speedup over eXpress without decreasing its accuracy. In addition, we show that the speedup achieved over eXpress increases as the data size and/or the number of iterations increases.

The rest of the paper is organized as follows. Section II explains algorithm of eXpress in detail and discusses its performance from several aspects. Section III discusses possible parallelization techniques for eXpress and presents our proposed approach. Section IV reports the experimental results to evaluate the performance of Par-eXpress. Section V presents related work and Section VI concludes the paper.

II. *eXpress* SOFTWARE

In this section, we present the algorithm underlying eXpress software (Section II-A) and discuss its performance (Section II-B).

A. Algorithm

eXpress [12] is a DNA/RNA sequence quantification tool, which performs probabilistic assignment of ambiguously mapped fragments by using the EM algorithm. Briefly, in the expectation step, the assignment probabilities are calculated based on current estimates of abundance parameters. Subsequently, the parameter estimates are updated in the maximization step. The parameters employed for this process include sequence bias, fragment length distribution, target abundances, and error transition probabilities. It should be noted that the convergence of the EM algorithm is not guaranteed when all these parameters are updated at each iteration [10]. Therefore, *auxiliary parameters* (*i.e.* fragment lengths and sequencing errors) are fixed at a certain point in order to guarantee convergence.

eXpress employs two types of iterations in its EM algorithm, namely *online* and *batch* iterations. In online iterations, parameters are updated after processing each fragment, while, in batch iterations, parameters are updated at the end of each iteration. As it is shown in [10], the best accuracy is achieved when the first iteration is online and the rests are batch. Once probabilistic assignment of fragments finishes, eXpress outputs parameter estimates and target abundances sorted by *bundles*[2].

Because we are focusing on parallelization, our discussion is centered on data processing structure of eXpress, and not the accuracy related algorithmic details. For the latter, please see original publications [10], [12].

eXpress takes two inputs, which are the target sequences (in Fasta format) and the aligned fragments (in SAM/BAM format). It employs a limited shared memory parallelization method by using 3 types of threads: 1) *data-loading thread* reads the fragments from disk and detects *hits* between target sequences and fragments 2) *data-processing thread* applies EM algorithm to learn the target abundance parameters, and 3) *auxiliary-parameter-updater thread* asynchronously updates the auxiliary parameters. Only one data-loading and one auxiliary-parameter-updater thread can be employed while multiple data-processing threads can be used.

The pseudo-code of eXpress tool is given as Algorithm 1 (as a serial execution for simplification). First, all target sequences are loaded into the memory (Line 1). Bundle set B is initialized to keep necessary info about bundles. Next, EM algorithm starts for a fixed number of iterations (Lines 3-21). For each fragment F loaded from the disk (Line 6), the set of target sequences that F can be mapped to (S_F), is detected (Line 7). Fragment F is further processed if S_F is not empty (Line 8). Otherwise, it is discarded and next fragment is read. The first iteration of EM algorithm is a special one, as mentioned. In this iteration, a bundle for the fragment F and its mapped target sequences is generated and merged with previously generated bundles, if possible. If there is no bundle that can be merged with, then it is added to the bundle set B (Line 10). Also, again only in the first iteration, the auxiliary parameters are learned. If $threshold_{start}$ number of fragments are processed, *auxiliary-parameter-updater* thread is started (Lines 11-12). This thread stops getting new fragments (*i.e.* updating auxiliary parameters) when $threshold_{stop}$ number

[2]A bundle is the transitive closure of all transcripts/target sequences sharing a mapped fragment. Bundles have presumable biological meaning since shared sequence suggests, but does not prove, shared homology, shared motif, and others. A sample bundle generation is illustrated in Figure 1.

Algorithm 1 eXpress Algorithm

```
Require: Target Sequence File (TSF), Alignment File (AF), number of iterations (M)
 1: $S \leftarrow load(TSF)$                          ▷ Load target sequences
 2: $B = \emptyset$                                   ▷ Initialize set of bundles
 3: $i = 0$
 4: while $i < M$ do
 5:     $N = 0$
 6:     for each Fragment $F$ in AF do
 7:         $S_F \leftarrow find\_hits(S, F)$
 8:         if $S_F \neq \emptyset$ then
 9:             if $i = 0$ then
10:                 $create\_or\_merge\_bundles(F, S_F, B)$
11:                 if $N > threshold_{Start}$ then
12:                     Start auxiliary-parameter-thread
13:                 else if $N < threshold_{Stop}$ then
14:                     Stop auxiliary-parameter-thread
15:                 $N = N + 1$
16:             Calculate assignment probabilities
17:             if $OnlineIteration$ then
18:                 Update parameters
19:     if $BatchIteration$ then
20:         Update parameters
21:     $i = i + 1$
22: Output results
```

of fragments are processed (Lines 13-14). The default values of $threshold_{start}$ and $threshold_{stop}$ are 1M and 5M, respectively. Assignment probabilities are calculated by the data-processing threads (*i.e.* expectation-step) (Line 16). If the type of the iteration is online, the parameters are updated after processing each fragment (Lines 17-18). But if it is a batch iteration, the parameters are updated at the end of the iteration (*i.e.* after processing all fragments) (Lines 19-20). The results are reported at the end of EM algorithm (Line 22).

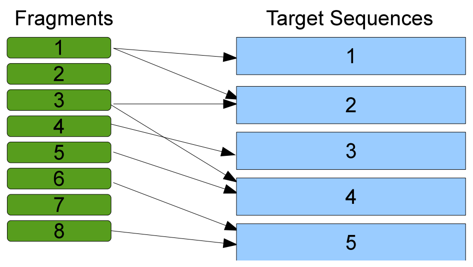
B. Performance Analysis

Now we discuss its performance according to following three criteria.

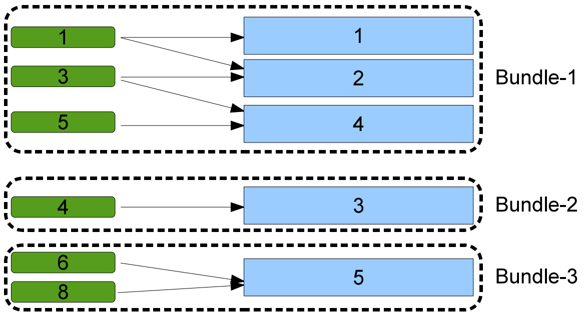
1) Memory Requirement: The target sequences are kept in the memory throughout the execution. However, each fragment is removed from the memory whenever it is processed. Thus, the memory allocation during the execution does not change and we can process any number of fragments with the same amount of memory consumption. In other words, eXpress software has constant memory requirement.

2) I/O Operations: Since the data is re-read at each iteration, I/O operations become one of the main bottlenecks limiting the performance. In addition, employing only one data-loading thread may not be suitable to run multiple data-processing threads. In other words, fragments can be processed much faster than they are loaded into the memory. Therefore, data-processing threads can stay idle for long time periods due to lack of fragments in the memory.

3) Parallel Scalability: eXpress has shared-memory parallelization and can employ multiple data-processing threads. However, due to *lock* operations and employing a single data-loading thread, it is not scalable at all. In order to analyze its parallel scalability empirically, we run eXpress with 2, 4, 8 and 16 threads, separately (one of them is data-loading thread). The dataset size was set to 4.6GB and the number of iterations was set to 2. The execution lasted 40 minutes in all cases. In other words, increasing number of data-processing threads is not increasing the performance of eXpress.



(a) Matching Fragments and Target Sequences



(b) Bundles

Fig. 1. Bundle Generation

III. Proposed Approach

A. Data Distribution

Now, we explore candidate distributed memory parallelization approaches for the algorithm of eXpress, and then discuss trade-offs among them. We can distribute target sequences and fragments across processes to parallelize the execution. There are 2 straightforward approaches to distribute the data:

1) **Distributing only target sequences**: In this method,

each process processes all fragments and a fraction of target sequences. This approach will have huge memory requirements if we keep all fragments in the memory. Thus, it cannot be applied for large-scale data analysis. Another option can be keeping only a single data chunk in the memory during execution. Then, the data has to be re-read from the disk at each iteration. In addition, if a fragment has hits from target sequences assigned to different processes, those processes need to synchronize and communicate with each other at each iteration to share their updated parameters. In the worst case, all processes may need to communicate with each other.

2) **Distributing only fragments**: Each process processes all target sequences and a fraction of fragments. In this method, we can keep all data in memory, since size of target sequences are usually much smaller than size of fragments. In addition, the available memory space can be increased easily by allocating more nodes. However, if fragments assigned to different nodes have hits for the same target sequences, then the corresponding processes need to communicate with each other at each iteration. That is, the network traffic and synchronization problems are valid for this approach, too.

The network traffic and synchronization related costs can be significant for both distribution methods. In order to overcome this problem, the fragments and target sequences of a bundle should be processed by a single process since bundles do not share any data by definition. Therefore, dividing the data according to bundles can be a reasonable approach for parallel processing. However, the content of each bundle can only be determined after processing all fragments, i.e., at the end of the first iteration. Another challenge is balancing the workload among processes. First, each fragment can have varying number of hits and the fragments with more hits will take longer time to be processed. Second, the size of bundles can vary dramatically. In order to understand this problem, we conducted a small experiment. We generated bundles with 10M fragments and 73660 target sequences and counted number of fragments in each bundle. The statistical results for this experiment are shown in Table I. We can see that the standard deviation of the number of fragments in bundles is extremely high. There is a single bundle, which covers 30% of all fragments while 58% of bundles has less than 10 fragments. Therefore, if we do not partition the bundles (*i.e.* each bundle is processed by a single process), there will be huge workload imbalance among processes.

We propose a bundle-based data distribution method that decreases the network traffic, but does not eliminate it totally. In our proposed approach, before distributing the data among processes, we first find the bundles and total

TABLE I
STATISTICAL VALUES FOR A SAMPLE DATA

Number of Target Sequences	73660
Number of Fragments	10M
Number of Bundles	23582
Average Number of Fragments	425
Standard Deviation of Fragments	19884
Number of Fragments in the Largest Bundle	3042142
Number of Bundles with less than 10 Fragments	13736

number of hits in each bundle in the first iteration. The parallelization of the first iteration is discussed in Section III-B. The bundles having more hits than a threshold are considered as *large bundles* that have to be processed by multiple processes. We set the threshold value to 1.5 times of average number of hits per process (*i.e.* N/P where N is total number of hits and P is number of processes). For each large bundle, we assign $\lceil N_B/(N/P) \rceil$ processes where N_B is the total number hits in the corresponding bundle. Once all large bundles are assigned, we distribute the rest of the bundles to the other processes such that each bundle is processed by only one process. Since bundles can have varying sizes and number of hits, we distribute these small bundles to the processes with a greedy approach such that the total number of hits for each process is balanced to the extent possible. Figure 2 illustrates how the data is distributed among processes.

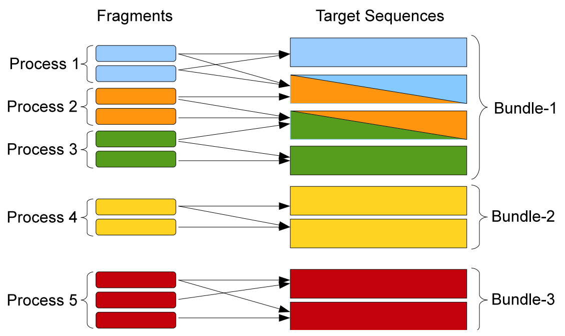


Fig. 2. Sample Data Distribution. The color of fragments and target sequences represents the process assigned for them. The target sequences that are processed by multiple processes are colored with multiple colors.

In our proposed approach, the processes that process small bundles do not need to communicate with each other and can perform their execution independently. However, processes assigned to the same large bundle have to synchronize and share their updated results at each iteration.

B. Parallelization of the First Online Iteration

The first iteration is crucial to achieve high accuracy [10] and has different processing structure than others. In the first iteration, bundles are generated and auxiliary parameters are learned. In addition, parameters are updated after processing each fragment and the updated parameters are used in the processing of the next fragment (*i.e.* online iteration). Thus, any parallelization attempt in this iteration will change the parameter estimates and eventually affect the accuracy. Since the scope of this work is to develop a distributed-memory parallel version of eXpress software without reducing its accuracy, we avoid any algorithmic change that affects the accuracy, even though it restricts our parallelization capability. Therefore, we use the original shared-memory parallelization in the first iteration and leave its distributed-memory parallelization as a future work. Additionally, we calculate the total number of hits and detect fragment ids in each bundle in order to calculate logical data distribution.

Algorithm 2 Parallel eXpress Algorithm

```
1: if $Process_{id}$ = 0 then
2:     Perform the threaded online iteration of eXpress
3:     Write the learned parameters and fragment ids of each bundle.
4:     Calculate the logical data distribution
5:     Notify the processes about data distribution
6: else
7:     Receive the bundle ids to be processed
8:     Load updated parameters
9: if Processing a large bundle then
10:     Load a fraction of the fragments of the assigned bundle
11: else
12:     Load all fragments of the assigned bundles
13: $i = 0$
14: while $i < iteration\_number$ do
15:     Process loaded fragments
16:     if Processing a large bundle then
17:         Share the updated parameters with the processes that process the same bundle
18:     $i = i + 1$
19: Combine partial results and report the final output
```

C. Parallel Execution Flow

Now, we explain entire parallel execution. Our proposed approach is given as Algorithm 2. Since we use original parallelization of eXpress (3 threads at maximum) in the first iteration, only Process-0 performs this critical iteration (Line 2). At the end of the first iteration, the learned parameters and fragment ids in each bundle are written to files (Line 3). Based on the number of hits in each bundle, Process-0 calculates the logical data distribution (Line 4) and notifies other processes (Line 5). During this process, other processes perform initialization actions and wait for results from Process-0. Note that, instead of writing fragment ids and parameters to files, we can also send the data directly to the processes. But we noticed that writing/loading parameters and bundle contents to/from the disk can be performed very fast[3]. In addition, by this approach, we can actually break the execution into two separate ones. So users can allocate only a single node for the first iteration and then allocate more nodes for the following iterations. In pay-as-you-go systems where users need to pay more as the number of allocated nodes increases (*e.g.* cloud computing systems), Par-eXpress's separated execution model will be useful to decrease the costs.

Once the tasks are distributed, all processes load the fragments of the bundles assigned to them (Lines 9-12). The processes processing a large bundle load only a fraction of the assigned bundle's fragments. For instance, if M processes will process a large bundle which has N fragments, then each process loads only N/M fragments. After loading fragments into memory, EM algorithm begins and runs for a certain number of iterations (Lines 13-19). The processes assigned for the same large bundle share their updated parameters each other at the end of each iteration (Line 17). Once all iterations are finished, the partial results of the processes are combined and final output is reported (Line 23).

IV. Experiments

In this section, we report results from a series of experiments we conducted to evaluate the accuracy and efficiency of Par-eXpress against eXpress.

A. Experimental Setup

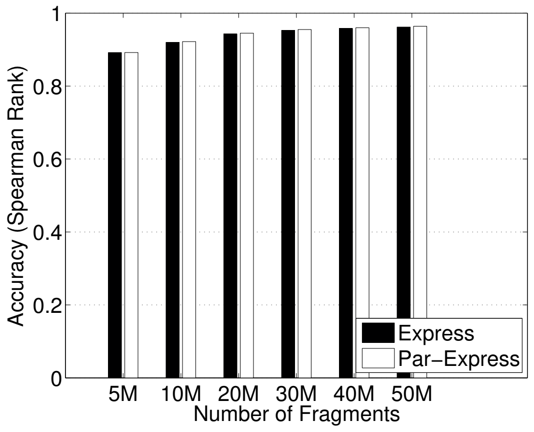
In our experiments, we used a cluster where each computing node has 8 cores 2.53 GHz Intel (R) Xeon (R) processor and 12 GB memory. The cluster has Lustre file system composed of 12x1TB disks across 4 storage nodes with 24GB memory. In our experiments with Par-eXpress, we used only one core per node. We used default parameters for eXpress. We employed the data used in [12].

B. Experimental Results

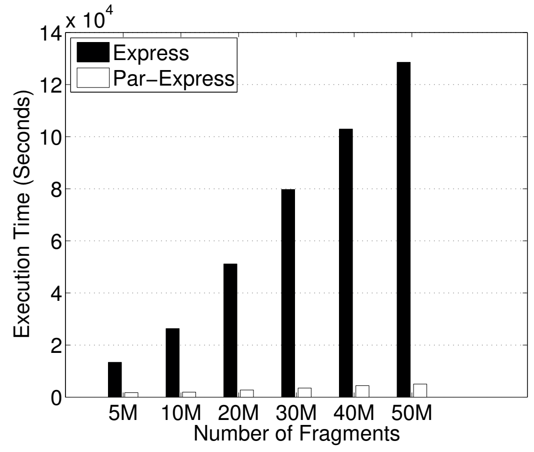
We now present experiments to evaluate accuracy and efficiency of Par-eXpress and compare it against eXpress from several aspects.

In this set of experiments, we run eXpress and Par-eXpress for 51 iterations (1 online and 50 batch iterations) with varying data sizes and compared their accuracy and

[3]For instance, when 10M fragments are processed with 8 processes, writing and loading the data needed for data distribution (i.e. bundle contents) takes 32 and 10 seconds, respectively.



(a) Accuracy Comparison



(b) Performance Comparison

Fig. 3. Accuracy comparison of eXpress and Par-eXpress with varying fragment sizes

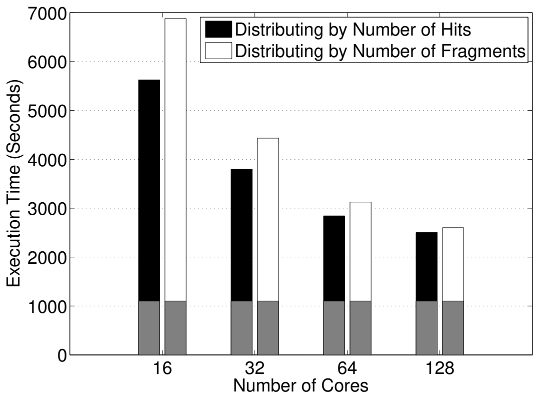


Fig. 4. Scalability of Par-eXpress and evaluation of our proposed data distribution metric. The gray part of the bars represent the execution time spent for the first iteration.

efficiency. We allocated 128 cores for Par-eXpress and varied number of fragments from 5M to 50M. The results are shown in Figure 3. In order to measure the accuracy, we calculated Spearman's Rank correlation against ground-truth results of the data. From Figure 3(a), we can see that Par-eXpress's accuracy results are slightly better than eXpress's results. This can be because eXpress's execution is not deterministic and small differences in accuracy are possible. The more important observation is that Par-eXpress did not cause any reduction in accuracy.

From Figure 3(b), we can see that Par-eXpress achieves high scalability with respect to dataset size. When dataset size is increased by 10 times, execution times of eXpress and Par-eXpress increase by 9.6x and 2.9x, respectively. In addition, speedup we achieve over eXpress increases from 7.7x to 25.5x. In other words, Par-eXpress becomes more efficient than eXpress as the dataset size increases. This is because eXpress reads the data at each iteration while Par-eXpress reads the data only once, which is after the first iteration.

In the next set of experiments, we evaluated the parallel scalability of Par-eXpress by varying number of allocated cores. We also compared our proposed data distribution method with a baseline method. In this baseline method, we used the number of fragments in bundles to estimate workload and also detect large bundles, instead of the number of hits. We set the number of iterations to 200 and the number of fragments to 10M and varied the number of processes from 16 to 128. The results are shown in Figure 4.

We can see that the number of hits is a better metric than the number of fragments in order to achieve workload balance. This is because processing fragments with more hits takes more time than the ones with fewer hits. The performance of Par-eXpress (distributing by number of hits) increases by 2.25x when computing power increases by 8x. There are three reasons that decrease the scalability of Par-eXpress. 1) The first iteration cannot be parallelized to preserve the accuracy, as explained in Section III-B. Thus, its execution time (shown in gray in the Figure) is same for all cases. In fact, if we ignore the execution time spent for the first iteration, the scalability of Par-eXpress increases to 3.2x. 2) As the number of processes increases, the overhead of communication increases since the large bundles will be shared by more processes. 3) Loading fragments does not scale well with the number of processes since fragments are not sorted nor indexed. In order to read a specific fragment, all previous fragments have to be scanned. For instance, the process assigned to process the last fragment has to scan all input data. If we just take account execution time of processes after loading fragments, the scalability of Par-eXpress increases to 5.1x.

In order to evaluate the parallel scalability of Par-eXpress from a different perspective, we varied the number of fragments and the number of processes together. We set the number of iterations to 200 and increased the number of fragments from 5M to 40M and the number of processes from 16 to 128. The results are shown in Figure 5. When the numbers of fragments and cores increased by 8x, the total execution time and the execution time after loading fragments increased by 1.7x and 1.2x, respectively. Considering the execution after the first iteration, as the data size increases, we can keep the execution time almost constant by increasing the computation power in the same order.

Overall, the parallelization after the first iteration has high parallel scalability. If an index that provides random access on alignment files (*i.e.* fragments) is built, the performance of Par-eXpress will increase even further.

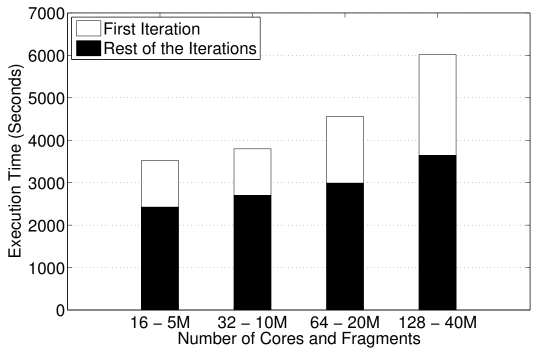


Fig. 5. Performance of Par-eXpress when computation power and data size are increased together. Upper white part of the bars represents the execution time spent in the first iteration and lower black part of the bars represents the execution time for the rest of the execution.

In our final set of experiments, we run eXpress and Par-eXpress with 5M fragments and varied number of iterations from 5 to 50. We allocated 128 cores for Par-eXpress. The results are shown in Figure 6. Par-eXpress's speedup over eXpress increases from 1.35x to 7.74x as the number of iterations increased from 5 to 50. eXpress needs to re-read data at each iteration and its execution time increases linearly as the number of iteration increases. On the other hand, Par-eXpress reads the data only twice (during the first iteration and also after assigning bundles) and keeps the data in the memory. Thus, after initial overhead of Par-eXpress, following iterations can be performed much faster. Overall, we can claim that Par-eXpress will be much faster than eXpress for the executions with higher number of iterations.

V. Related Work

Our work falls in the general area of *sequence quantification*, which has received significant attention. Bayesian

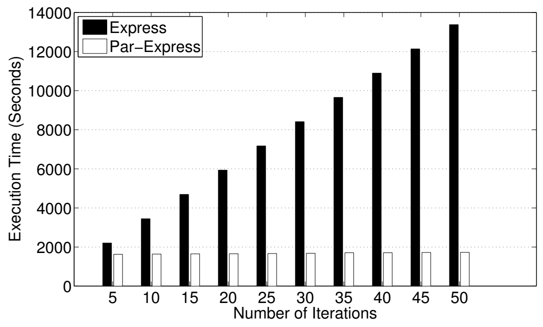


Fig. 6. Efficiency comparison in varying iterations.

inference [6] and Expectation Maximization [5], [7], [13], [14] are among popular approaches for this problem. Most of the research aims to develop new algorithms to achieve higher accuracy. In comparison, our goal is to parallelize an existing approach without decreasing its accuracy.

Most of the popular quantification tools such as eXpress, Cufflink [14] and RSEM [5], require alignment of fragments by an external alignment tool (*e.g.*, Bowtie2 [3], TopHat2 [2]) before quantification. There are also studies that do not require this alignment step, and thereby increase the efficiency, such as kallisto [1], Sailfish [9] and RNA-Skim [17]. But our work runs on aligned fragments and supports distributed memory parallelization to increase efficiency.

There are also a number studies that employ parallelization to increase efficiency. EMSAR [4], RSEM [5], Salmon [8] and Sailfish [9] support only shared memory parallelization. eXpress-D [11] employs Spark [15] for the distributed-memory parallelization of eXpress with slight changes in the original algorithm. eXpress-D first employs a shared-memory parallel pre-processing step, in which fragment hits are detected, input data is re-organized, and written in *Protocol Buffer* format in order to decrease memory requirements of the following steps. Before parallel processing, the data needs to be uploaded to a distributed file system, such as Hadoop Distributed File System or Amazon S3. In parallel execution of eXpress-D, all nodes synchronize and share their results at each iteration. One significant limitation of eXpress-D is that fragments with insertions or deletions cannot be processed. Therefore, it cannot be used in many genomic research areas such as cancer. Our work has the following advantages over eXpress-D. 1) The original input files (in SAM/BAM and FASTA formats) can be used directly (i.e. No new input data is generated). 2) We decrease the number of nodes to be synchronized by distributing data more carefully. 3) There is no restriction on the content of the input data.

That is, fragments with insertions or deletions can also be processed. 4) eXpress-D requires a distributed file system and a Spark installation, which are not popular among bioinformaticians. However, Par-eXpress requires only MPI library, and can be easily used on a typical batch-request high performance computing installation, which is available to many bioinformaticians.

VI. Conclusion

Probabilistic assignment of ambiguously mapped fragments is common yet time-consuming procedure for many sequencing experiments. Therefore, in this work, we developed a distributed-memory parallel version of eXpress software. We have discussed different parallelization approaches for this problem and proposed an approach, which preserves the accuracy of original eXpress software while achieving high efficiency. We used multi-threaded execution of eXpress in the first iteration to learn the parameters effectively. The actual distributed-memory parallel execution begins after the first iteration. We proposed a data distribution method, which decreases the network traffic and synchronization operations.

The main observations from our extensive evaluation are as follows. First, Par-eXpress is able to achieve significant speedup over eXpress without decreasing its accuracy. Second, its performance can increase even further with an index that provides random access on genomic files. Finally, the speedup achieved by Par-eXpress increases as the number of iterations and/or data size increases. Therefore, it can be used to process large scale genomic data.

Our future work will focus on assigning fragments of multiple genomic data files in parallel, in order to efficiently conduct multi-sample experiments.

References

[1] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic rna-seq quantification. *Nature biotechnology*, 34(5):525–527, 2016.

[2] Daehwan Kim, Geo Pertea, Cole Trapnell, Harold Pimentel, Ryan Kelley, and Steven L Salzberg. Tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biol*, 14(4):R36, 2013.

[3] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357–359, 2012.

[4] Soohyun Lee, Chae Hwa Seo, Burak Han Alver, Sanghyuk Lee, and Peter J. Park. Emsar: estimation of transcript abundance from rna-seq data by mappability-based segmentation and reclustering. *BMC Bioinformatics*, 16(1):278, 2015.

[5] Bo Li and Colin N Dewey. Rsem: accurate transcript quantification from rna-seq data with or without a reference genome. *BMC bioinformatics*, 12(1):323, 2011.

[6] Naoki Nariai, Kaname Kojima, Takahiro Mimori, Yukuto Sato, Yosuke Kawai, Yumi Yamaguchi-Kabata, and Masao Nagasaki. Tigar2: sensitive and accurate estimation of transcript isoform expression with longer rna-seq reads. *BMC genomics*, 15(Suppl 10):S5, 2014.

[7] Marius Nicolae, Serghei Mangul, Ion I Măndoiu, and Alex Zelikovsky. Estimation of alternative splicing isoform frequencies from rna-seq data. *Algorithms for molecular biology*, 6(1):9, 2011.

[8] Rob Patro, Geet Duggal, and Carl Kingsford. Salmon: accurate, versatile and ultrafast quantification from rna-seq data using lightweight-alignment. *bioRxiv*, page 021592, 2015.

[9] Rob Patro, Stephen M Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from rna-seq reads using lightweight algorithms. *Nature biotechnology*, 32(5):462–464, 2014.

[10] Adam Roberts. *Ambiguous fragment assignment for high-throughput sequencing experiments*. University of California, Berkeley, 2013.

[11] Adam Roberts, Harvey Feng, and Lior Pachter. Fragment assignment in the cloud with express-d. *BMC bioinformatics*, 14(1):358, 2013.

[12] Adam Roberts and Lior Pachter. Streaming fragment assignment for real-time analysis of sequencing experiments. *Nature methods*, 10(1):71–73, 2013.

[13] Hong Sun, Shuang Yang, Liangliang Tun, and Yixue Li. Iaoseq: inferring abundance of overlapping genes using rna-seq data. *BMC bioinformatics*, 16(Suppl 1):S3, 2015.

[14] Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and abundance estimation from rna-seq reveals thousands of new transcripts and switching among isoforms. *Nature Biotechnology*, 28(5):511, 2010.

[15] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[16] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[17] Zhaojun Zhang and Wei Wang. Rna-skim: a rapid method for rna-seq quantification at transcript level. *Bioinformatics*, 30(12):i283–i292, 2014.