IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 27, 2019, accepted October 21, 2019, date of publication November 4, 2019,
date of current version November 14, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2951284

GPU Implementation of Pairwise Gaussian Mixture Models for Multi-Modal Gene Co-Expression Networks

BENJAMIN T. SHEALY[1,*], (Student Member, IEEE), JOSH J. R. BURNS[2,*], MELISSA C. SMITH[1], (Senior Member, IEEE), F. ALEX FELTUS[3], AND STEPHEN P. FICKLIN[2]

[1]Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634, USA
[2]Department of Horticulture, Washington State University, Pullman, WA 99163, USA
[3]Department of Genetics and Biochemistry, Clemson University, Clemson, SC 29634, USA

Corresponding author: Stephen P. Ficklin (stephen.ficklin@wsu.edu)

*Benjamin T. Shealy and Josh J. R. Burns are co-first authors and have contributed equally to this work.

This work was supported in part by the National Science Foundation Award #1659300 "CC*Data: National Cyberinfrastructure for Scientific Data Analysis at Scale (SciDAS)."

ABSTRACT Gene co-expression networks (GCNs) are widely used in bioinformatics research to perform system-level analyses of organisms based on the pairwise correlation between all expressed genes. For large datasets which contain samples from multiple sources, gene pairs can exhibit multiple modes of co-expression which confound typical correlation approaches. A clustering method such as Gaussian Mixture Models (GMMs) may be used to separate the modes of each gene pair in an unsupervised manner, prior to computing the correlation of each mode. However, pairwise clustering significantly increases the computational cost of constructing a GCN, as several clustering models must be evaluated for each gene pair, and the number of gene pairs grows rapidly with the number of genes. In this paper, we present a heterogeneous, high-throughput multi-CPU/GPU software package for multi-modal GCN construction, implemented in version 3 of the Knowledge Independent Network Construction (KINC) software. We determine the optimal values for several execution parameters of the GPU implementation, and we benchmark our CPU and GPU implementations for up to 8 CPUs/GPUs. Our GPU implementation achieves a 167x speedup over the corresponding CPU implementation, as well as a 500x speedup over KINCv1.

INDEX TERMS Bioinformatics, Gaussian mixture model, gene co-expression network, gene expression matrix, GPU computing, high-performance computing, high-throughput computing.

I. INTRODUCTION

High-throughput gene expression profiling technologies such as microarrays, and more recently RNA-Seq, have enabled researchers to examine molecular systems at unprecedented resolution. With these technologies, it is possible to measure individual molecules in order to quantify gene output and detect subtle DNA sequence changes in billion of letters of genetic code. Gene output across the tree of life is directly connected to traits of importance that directly affect human health, food security, response to the climate, and other grand challenges. Thus, understanding the relationship between gene output data and traits of importance has broad impact for biological research. With high-throughput gene measurements, it is easier for the biologist to ask questions at the systems-level and apply the scientific method while accounting for the inherent complexity of biological phenomena. These high-throughput measurement techniques, especially RNA-Seq, have also led to a geometric increase in the amount of new data. In the last decade, the NCBI Sequence Read Archive (SRA) repository alone has received more than 30 petabytes of sequence data [1] with much of that data derived from experiments designed to measure gene expression under a variety of biological conditions. This increase in data now affords the opportunity, through combined datasets, to gain new insights beyond the scope of the original experiments. Moreover, decreased costs of DNA sequencing enables discrete experiments to include

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Alawneh.

measurements for multiple experimental conditions. In both cases, the result is increasingly multidimensional datasets where intrinsic and extrinsic variation are confounding. Accelerated computational approaches are needed to address challenges of analyzing these high dimensional data, and to tease out relationships between genes and the biological function under which they are expressed.

In this work we describe a heterogeneous, high-throughput, multi-CPU/GPU implementation that addresses the challenges of analyzing high-dimensional gene expression data, and models that data in a graph structure known as a gene co-expression network (GCN). We developed this software and tested it on a cluster equipped with NVIDIA P100 and V100 GPUs. Here, we provide descriptions of the data consumed and produced by this application, as well as the algorithms used by it. We examine and optimize several execution parameters of our implementation, and we measure its overall speedup over previous work for several hardware resource sets. We also discuss two key performance issues with the (naive) GMM kernel – memory coalescing and workload balance – and we measure the effect of the optimizations that we applied to mitigate these bottlenecks.

II. BACKGROUND

A. GENE EXPRESSION MATRIX

A gene expression matrix (GEM) is a $n \times m$ dataset with n rows of genes and m columns of samples. Each element in the GEM is a numeric value that represents the expression level of a particular gene in a particular sample. Higher relative gene expression often correlates with an increase in the functional activity of that gene. The two most common technologies for measuring gene expression are microarray (an older chip-based DNA hybridization assay) and RNA-Seq (a newer DNA-sequencing approach that often yields gigabytes of data for a single sample). The expression-level value is a positive real value which can be represented as a florescence intensity measure (from microarrays), a raw integer count, or a normalized count either as Fragments Per Kilobase of transcript per Million mapped reads (FPKM), or Transcripts Per Kilobase Million (TPM) (from RNA-Seq). Additionally, it is common practice to transform the GEM using an element-wise log2 transformation. As a result, genes which were not expressed or insufficiently measured (count $= 0$) are transformed to $-\infty$, and thus represented in the transformed matrix as a missing value (NAN).

B. GENE CO-EXPRESSION NETWORKS

A gene co-expression network (GCN) is a graph in which each node is a gene and an edge exists between two nodes if there is significant correlation of expression (co-expression) between those two genes. A GCN can be constructed from a GEM by computing the similarity between each gene pair using a similarity measure such as Pearson, Spearman, Kendall Tau, Mutual Information, or the biweight midcorrelation [2]. The resulting similarity matrix is converted into an adjacency matrix by applying a significance threshold (calculated or ad-hoc, user-specified). This two-step process of similarity detection and thresholding is the most common approach to GCN construction and is implemented by a variety of software packages.

Many genes are multifunctional, meaning they participate in more than one biological process. Therefore, as gene expression datasets become larger and more diverse, they become increasingly multidimensional. Multifunctional genes expressed under different experimental conditions can exhibit multi-modal expression patterns, which violate the assumptions of most correlation tests. Using conventional pairwise correlation analysis, in this case, results in more false correlations being included and more true correlations being excluded. Instead, each mode within the gene pair should be evaluated separately, producing a *multi-modal gene co-expression network* in which two nodes (genes) can have multiple edges between them. In many cases, theses modes can be correlated to specific biological conditions (as shown in Figure 2). This approach is implemented in KINC version 1 (v1) [3], which uses Gaussian Mixture Models (GMMs) for pairwise clustering. KINCv1 has been used to find condition-specific gene relationships (i.e. graph edges) in large and diverse cancer datasets [4], [5] and post-harvest scald in apples [6].

Using the GMM approach, KINCv1 is able to capture multi-modal relationships between genes, but it is also much more computationally expensive than traditional co-expression network construction approaches for two reasons. First, every gene pair must be evaluated using the GMM clustering algorithm, because it is unknown which gene pairs will have multiple modes. Second, the GMM approach must be given the number of clusters or "components" beforehand, and it is unknown how many clusters each gene pair contains. Several clustering models must be computed and compared for each gene pair. As a result, the runtime of KINCv1 increases rapidly with the number of genes ($\mathcal{O}(n^2)$). For a large dataset (on the order of 50,000 genes), constructing a multi-modal GCN with KINCv1 can take weeks or even months, even on a computing system with thousands of CPU cores.

C. GAUSSIAN MIXTURE MODELS

A Gaussian mixture model (GMM) is a probabilistic model which is commonly used for cluster analysis [8], [9]. It assumes that data is distributed as a Gaussian mixture with K components

$$p(y|x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(y; \mu_k, \Sigma_k),$$

where $\mathcal{N}(\mu, \Sigma)$ is the multivariate normal distribution

$$\mathcal{N}(\mu, \Sigma) = (2\pi)^{-\frac{k}{2}} \det(\Sigma)^{-\frac{1}{2}} e^{-\frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)}.$$

A GMM is fit to a dataset $X = \{x_1, \ldots, x_N\}$ using the Expectation-Maximization (EM) algorithm, which is an iterative algorithm with two steps in each iteration.

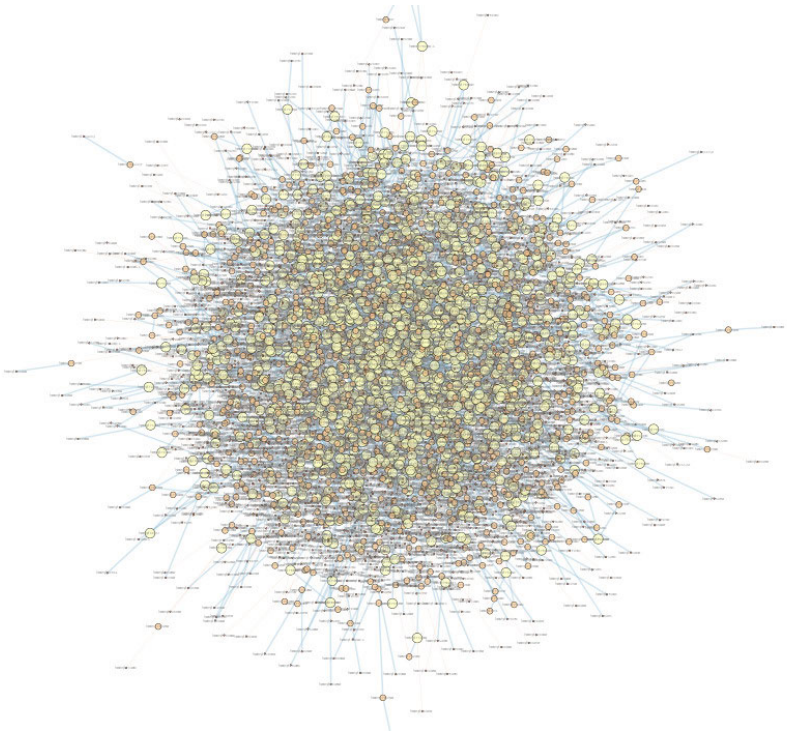

FIGURE 1. Gene co-expression network for *Saccharomyces cerevisiae* (yeast).

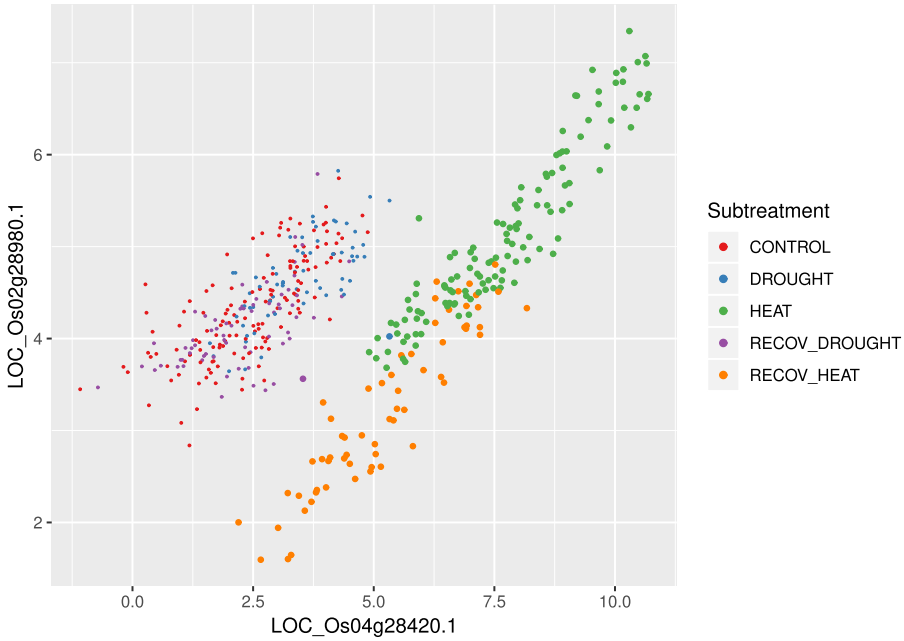

FIGURE 2. Pairwise correlation between two genes in *Oryza sativa* (rice). Gene expression measurements from 475 samples were taken under experimental conditions where rice plants were exposed to heat, heat recovery, drought and drought recovery [7]. Points represent samples and are colored according to the experimental treatment to which they were exposed.

In the expectation step (E-step), the conditional probabilities of each data point for each GMM component are computed as

$$\gamma_{ik} = p(x_i|\pi_k, \mu_k, \Sigma_k) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}.$$

In the maximization step (M-step), the parameters of the mixture model are re-computed as the maximum likelihood estimate of the data

$$n_k = \sum_{i=1}^{N} \gamma_{ik},$$

$$\pi_k = n_k/N,$$
$$\mu_k = \frac{1}{n_k}\sum_{i=1}^{N}\gamma_{ik}x_i,$$
$$\Sigma_k = \frac{1}{n_k}\sum_{i=1}^{N}\gamma_{ik}(x_i-\mu_k)(x_i-\mu_k)^T.$$

The update equation for the covariance Σ_k is for the *full covariance* model, in which the covariance is not constrained in any way. Alternatively, the covariance can be constrained to be spherical, diagonal, or constant across all components. In our case the full covariance is the most appropriate option as it can model clusters with arbitrary shape and orientation.

The E-step and M-step are repeated for a fixed number of iterations or until the log-likelihood of the model converges:

$$|\mathcal{L}_t - \mathcal{L}_{t-1}| < \epsilon,$$
$$\mathcal{L}(x|\theta) = \sum_{i=1}^{N}\sum_{j=1}^{K}\pi_j\mathcal{N}(x_i|\mu_j, \Sigma_j).$$

Furthermore, the model parameters can be initialized in a number of different ways. We use the following strategy:

1) Initialize each mixture weight to $1/K$
2) Initialize the mixture means using K-means clustering on X
3) Initialize each mixture covariance to the identity matrix

The final output labels are determined by the conditional probabilities computed in the expectation step

$$y_i = argmax(\gamma_{ik}, k).$$

Like most clustering algorithms, GMMs must be provided with the number of clusters that are known to exist in the input data. If the number of clusters is not known, several models must be fit to the input data, one model for each possible value of K, and the best model must be selected. There is no objective way to determine the "best" model for unlabeled data, but there are several model selection metrics which are commonly used, including AIC, BIC, and ICL [10],

$$AIC = 2\,p - 2\ln(\mathcal{L}),$$
$$BIC = \ln(N)p - 2\ln(\mathcal{L}),$$
$$ICL = \ln(N)p - 2\ln(\mathcal{L}) + 2\,E,$$

where N is the number of samples, p is the number of model parameters, $\mathcal{L}$ is the likelihood of the model, and E is an entropy term. In general, these metrics aim to maximize goodness-of-fit (measured by the log-likelihood) while also minimizing model complexity (measured by the number of parameters). For each of these metrics, a lower score is better.

D. SIMILARITY MATRIX CONSTRUCTION

Given a gene expression matrix $\mathbb{R}^{n\times m}$ (that is, containing n genes and m samples), we compute the multi-modal similarity matrix $\mathbb{R}^{n\times n\times K}$, where $S_{i,j}$ is the similarity between genes i and j. $S_{i,j}$ may contain up to K "modes" or clusters, and so $S_{i,j,k}$ is the similarity of the k-th cluster identified in the pairwise data of genes i and j.

Since S is symmetric, and since the diagonal elements of S correspond to comparing genes to themselves which is not useful, only the lower (or upper) triangle of S needs to be computed. Therefore, the construction of S essentially consists of iterating through the lower triangle of S and determining for each element (1) the number of clusters, (2) the cluster labels, and (3) the correlation of each cluster. The total number of pairwise comparisons that must be performed for n genes is $\frac{n(n-1)}{2}$; for example, a gene expression matrix with $1,000$ genes yields $499,500$ pairwise comparisons. A high-level description of similarity matrix construction is given in Algorithm 1.

Algorithm 1 Multi-Modal Similarity Matrix Construction

```
procedure Similarity(E, n, m, K_max)
    S ← ℝ^{n×n×K_max}
    for i ∈ 1..n do
        for j ∈ 1..i do
            X ← FetchPair(E, i, j)
            X ← RemoveOutliers(X)
            K ← GMM(X, K_max)
            for k ∈ 1..K do
                X_k ← RemoveOutliers(X_k)
                S_{i,j,k} ← Correlation(X_k)
            end for
        end for
    end for
    return S
end procedure
```

III. RELATED WORK

There exist other packages for constructing gene co-expression networks, including WGCNA [11], petal [12], FastGCN [13], and KINCv1 [4], [14]. KINC is the only GCN software that performs pairwise clustering in order to detect multi-modal gene pair relationships. KINCv1 can be run on a single CPU or on a CPU cluster in a high-throughput manner. We refer to our multi-CPU/GPU implementation in this work as KINCv3 in order to distinguish it from KINCv1.

Gaussian mixture models are used for a wide variety of tasks, including background subtraction [15] and speaker verification [16], [17]. Additionally, several GMM implementations for these domains have been published for GPUs [18], [19] and FPGAs [20], [21], as well as general-purpose GMM implementations for GPUs [22] and FPGAs [23]. Arguably the most similar work to ours is [19], in which the authors accelerated a GMM-based background subtraction algorithm by processing each pixel-wise GMM in a separate thread, in the same way that the KINCv3 GMM kernel devotes a thread to each gene pair. Many of the techniques used by [19], including pinned host memory, memory coalescing, and multiple CUDA streams, are also used here.

To the best of our knowledge, KINCv3 is the first GPU implementation of pairwise GMMs for gene expression data.

Additionally, many of the FPGA implementations show promising results, such as the 517x speedup achieved by [21]. Considering that FPGAs can outperform their GPU counterparts in some cases, such as data-intensive applications, by devoting hardware resources in a more fine-tuned manner, it may be worthwhile in the future to explore an FPGA implementation for the GMM kernel in KINC.

IV. IMPLEMENTATION

A. PAIRWISE SAMPLE EXTRACTION

The first step in processing a gene pair (i, j) is to extract the pairwise samples for genes i and j from the expression matrix. Due to the occurrence of missing values in the expression matrix, any pairwise sample with a missing value in either gene must be excluded for that gene pair. Additionally, a pairwise sample can be excluded for other reasons, such as if either gene expression falls below a user-specified threshold, or if either expression is identified as an outlier prior to clustering or correlation analysis. As a result, the number of pairwise samples can vary greatly for each gene pair.

B. PAIRWISE GMM

General-purpose GMM implementations such as mixmod [24] and the `sklearn.mixture` module in scikit-learn [25] are designed to work with an arbitrary number of input dimensions. Since KINC only considers pairwise GMMs, for which the input data is two-dimensional, we use this assumption to simplify several aspects of the GMM algorithm. In particular, all matrix and vector operations are simplified by loop unrolling, and the matrix inverse required in the E-step is computed directly in the two-dimensional case in lieu of using the Cholesky decomposition. We incorporated these optimizations into the CPU implementation of KINCv3, which contributed to a significant performance improvement over KINCv1 as discussed in the Results section.

C. MODEL INITIALIZATION

The parameters of each GMM are initialized using K-means clustering, which can be considered a special case of GMM that assumes a uniform mixture proportion and identity covariance for each component. The K means are themselves initialized randomly from the input data, and the algorithm is run partially (up to 20 iterations) rather than until full convergence. Additionally, only one initialization is used for each value of $K \in \{1, 2, 3, 4, 5\}$, even though it is best practice to compute multiple GMMs with different initializations, due to the large number of GMMs that must be computed at scale.

These initialization choices are a trade-off between improving results and minimizing runtime, as the initialization of model parameters can significantly impact the clustering results, but any additional work per gene pair increases total runtime significantly because the total runtime complexity is $\mathcal{O}(n^2)$ with respect to the number of genes.

D. MODEL SELECTION

Since the number of clusters in each gene pair is unknown, several GMMs are computed for each gene pair and the best GMM is selected using a model selection criterion. We use $K \in \{1, 2, 3, 4, 5\}$, a reasonable range for the possible number of modes in a gene pair based on experience, and we use the Integrated Computed Likelihood (ICL) to select the best model. Thus five GMMs must be computed for each gene pair, and the model with the lowest ICL score is selected for the given gene pair.

E. GPU IMPLEMENTATION

Since KINC must compute many small GMMs, we implemented a CUDA kernel[1] in which each thread performs the entire clustering and model selection process for a single gene pair. That is, each thread computes five GMMs and selects the GMM with the lowest ICL score. We implemented separate kernels for pairwise sample extraction, outlier removal, and correlation analysis, such that each kernel is executed sequentially as shown in Algorithm 1. These kernels operate in the same way as the GMM kernel in that each thread processes a gene pair, however they are not examined in detail here as they represent only 1-2% of total GPU activities reported by `nvprof`.

All kernel executions and memory transfers are performed asynchronously with respect to the host, and all memory transfers use pinned host memory to maximize memory bandwidth between host and device memory. Memory transfers, which are required for the expression matrix at the beginning and the input and output data for each work block, represent approximately 0.1% of total GPU activities, so they are also not examined further here.

F. MULTI-GPU EXECUTION WITH ACE

The Accelerated Compute Engine (ACE) is a C++ library for writing high-throughput scientific applications [26]. KINCv3 uses ACE as a back-end for parallel execution, which allows KINC to take advantage of distributed heterogeneous computing environments such as a GPU cluster.

ACE provides a parallelization framework (based on MPI) which designates the first process as the *master* process, which distributes and collects *work blocks*, and all other processes as *worker* processes, which process work blocks. A work block is a collection of *work items*, and a work item in KINC is a single pairwise comparison. Given the total number of work items N and a work block size B, the master divides the work into N/B work blocks of B work items (gene pairs) each, and distributes work blocks dynamically to the worker processes. Each worker receives a few work blocks at the beginning, and when a worker finishes processing a work block, it returns a *result block* with the results for each work item to the master, at which point it receives another work block from the master.

[1]The reader is referred to the literature for background information on the CUDA programming model.

This dynamic work assignment is designed to address two important sources of variation. First, work blocks generally do not require the same amount of time to process as some gene pairs may not have enough pairwise samples to perform cluster analysis. Second, in a heterogeneous environment each worker may be equipped with different GPU hardware, or may not have a GPU at all, which means that some workers will process work blocks more quickly than others. Since each worker receives work blocks only as quickly as it can process them, KINC can provide workload balance despite these variations.

A work block consists of the index of the first gene pair and the number of gene pairs to process. However, the worker must be able to determine the pairwise index (i, j) of a gene pair given its work item index, so that it can retrieve the corresponding rows for each gene in the gene expression matrix. This procedure is described in Algorithm 2.

Algorithm 2 Pairwise Index Method

```
procedure PairwiseIndex(i)
    p ← 0
    x ← 0
    while p + x ≤ i do
        p ← p + x
        x ← x + 1
    end while
    return (x, i − p)
end procedure
```

The multi-GPU implementation introduces an additional level of parallelism within each work block. Whereas the CPU-enabled worker simply processes work items sequentially, the GPU-enabled worker offloads the processing of work items to the GPU through a series of kernel executions. That is, given a work block of B work items and a *global work size* G (also the CUDA grid size), the GPU-enabled worker executes B/G kernels sequentially to process the entire work block. The threads in a kernel execution are also organized into thread blocks of size L, which we call the *local work size* (also the CUDA block size).

V. PERFORMANCE OPTIMIZATIONS

In our GMM kernel, each thread individually computes several GMMs, which means that this kernel is both compute-intensive and memory-intensive. The performance limitations of the GMM kernel can be summarized by two issues: global memory congestion and workload imbalance. We discuss each of these issues and our solutions to them in the following sections.

A. MEMORY COALESCING

Each thread in the GMM kernel uses several workspace arrays in global memory for intermediate data. Although it is unlikely that the global memory usage of this kernel can be reduced further, it is possible to improve the *memory access patterns* by manipulating the layout of the workspace arrays in global memory. Each workspace array contains the workspace for every thread in the grid, such that each thread uses a contiguous block of memory within the array. In cases where workspace elements need not be contiguous in memory, these arrays could be refactored such that each thread uses strided indexing. That way, when all threads in a warp access their respective elements in the same workspace array, the elements will be contiguous in memory and will be fetched with fewer global memory accesses than in the naive implementation. This technique is referred to as *memory coalescing* and it is a common approach to maximizing global memory throughput on GPUs. It was also easy to implement, since only the loop indices needed to be modified rather than the array accesses themselves.

We should note that strided indexing can guarantee contiguous memory accesses only in the case of two-dimensional arrays (each thread with it's own one-dimensional array). For higher-order arrays, the degree of memory coalescing depends on how the array is accessed by the threads. The GMM kernel contains one such higher-order array: $\Gamma \in \mathbb{R}^{N \times K}$ (*gamma*), the array of conditional probabilities. Since *gamma* is always accessed at the same index by all threads, relative to each thread's offset, the memory accesses for *gamma* will be contiguous. In other words, *gamma* is effectively used like a one-dimensional array, so it will be coalesced as such.

A related but separate consideration is the arrangement of $\theta = \{(\pi, \mu, \Sigma)_k\}$ (*theta*), the array of GMM components. We observed a significant performance benefit by arranging *theta* as a structure-of-arrays instead of an array-of-structures. This step increases memory coalescing by itself but it is also crucial in enabling memory coalescing via strided indexing. Since the structure-of-arrays pattern was already implemented in both the CPU and naive GPU implementations, we do not measure it's individual benefit here.

B. WORKLOAD BALANCING

The GMM kernel is very large and contains many loops and branches. Although there are many *if* statements, there are no *else* statements that would trigger additional passes, so branching does not contribute to control divergence. The loops can be divided into three primary categories: loops over the number of EM iterations T, the number of clusters K, and the number of samples N. The number of clusters is constant across all threads, so it does not contribute to control divergence. The number of iterations will vary as some threads converge before others (early stopping). The number of samples can vary drastically for each thread, as shown in Figure 3, due to the occurrence of missing values in the input data, which are excluded from the clustering analysis. As a result, threads with fewer samples must wait on threads with more samples during any loop over N. Therefore the primary source of control divergence is *workload imbalance* due to variation in (1) the number of EM iterations and (2) the number of extracted pairwise samples for each gene pair.

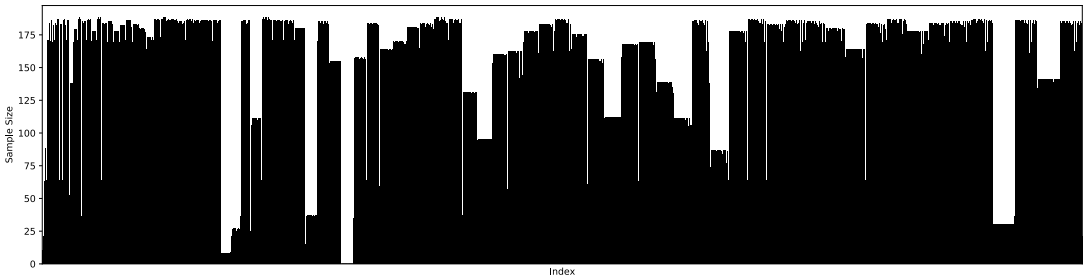

FIGURE 3. Sample sizes for the first 4096 gene pairs in the yeast dataset, sorted by pairwise index.

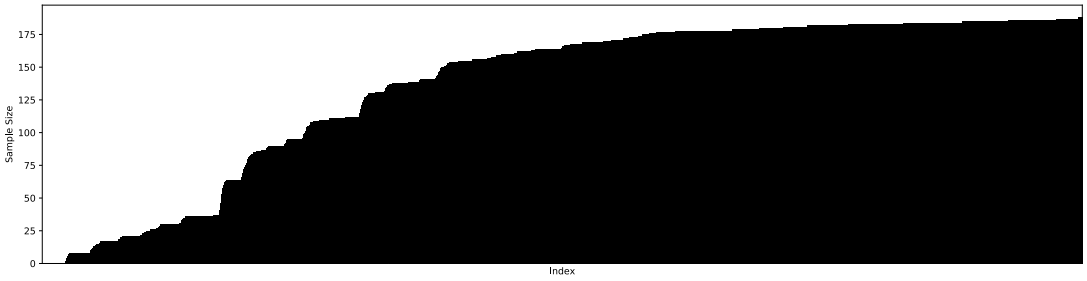

FIGURE 4. Sample sizes for the first 4096 gene pairs in the yeast dataset, sorted by sample size.

Workload imbalance due to the number of iterations could be eliminated by simply removing the early stopping condition such that all threads perform the maximum number of iterations. However, this change also increases global memory congestion as threads perform useless work, leading to an overall performance loss. Therefore we found that this imbalance is unavoidable.

However, we realized that the second source of imbalance could be reduced by mapping gene pairs to threads according to sample size, such that contiguous threads within a warp have similar sample sizes. The sample size for each gene pair is not known until after the pairwise sample extraction step, so we only sort G pairs at a time, as shown in Figures 3 and 4. We perform an indirect sort (argsort) on the sample size array and use the resulting indices to map threads to gene pairs according to sample size. This approach is vastly superior to a direct sort because only one sort is required on the argsort array, whereas a direct sort would require every input and output array to be sorted in order to ensure that the output arrays are ordered by pairwise index.

There is an interesting relationship between these two optimizations, strided indexing and pairwise sorting. When threads are mapped to different inputs in order to maximize workload balance, their memory accesses are also rearranged and may become more spread out depending on the argsort, which in turn may detract from the performance benefit provided by strided indexing. Indeed, we found that it is better not to apply the sorting mechanism to the work arrays, since it doesn't matter how the work arrays are mapped to threads and re-ordering them would only decrease memory coalescing. Therefore, we apply the sorting mechanism only to the arrays for which it is required, which are the input and output arrays, in order to minimize its impact on memory coalescing.

VI. EXPERIMENTAL SETUP

For all of our experiments we used a *Saccharomyces cerevisiae* (yeast) RNA expression dataset, obtained from the NCBI GEO database [27], which contains 188 samples and 7,050 genes. We ran our experiments using NVIDIA P100 and V100 GPUs on the Palmetto cluster at Clemson University. The P100 nodes are equipped with Intel Xeon E5-2680v4 CPUs, and the V100 nodes are equipped with Intel Xeon 6148G CPUs. We used the CPU performance results from the P100 nodes as the baseline for all GPU performance comparisons, as we found that the CPU performance was nearly identical between the P100 and V100 nodes.

TABLE 1. KINCv3 multi-GPU execution parameters.

Parameter	Description	Default Value
Worker threads	host threads per GPU	1
Work block size	gene pairs per work block	32768
Global work size	threads per CUDA grid	4096
Local work size	threads per CUDA block	32

There are a number of parameters that control the execution of the multi-GPU implementation. We evaluated the effect of each parameter on performance in order to determine its optimal value. We used the default values shown in Table 1 for each parameter when it was not being varied.

To demonstrate the effect of the two GMM kernel optimizations, as well as their combined effect, we benchmarked

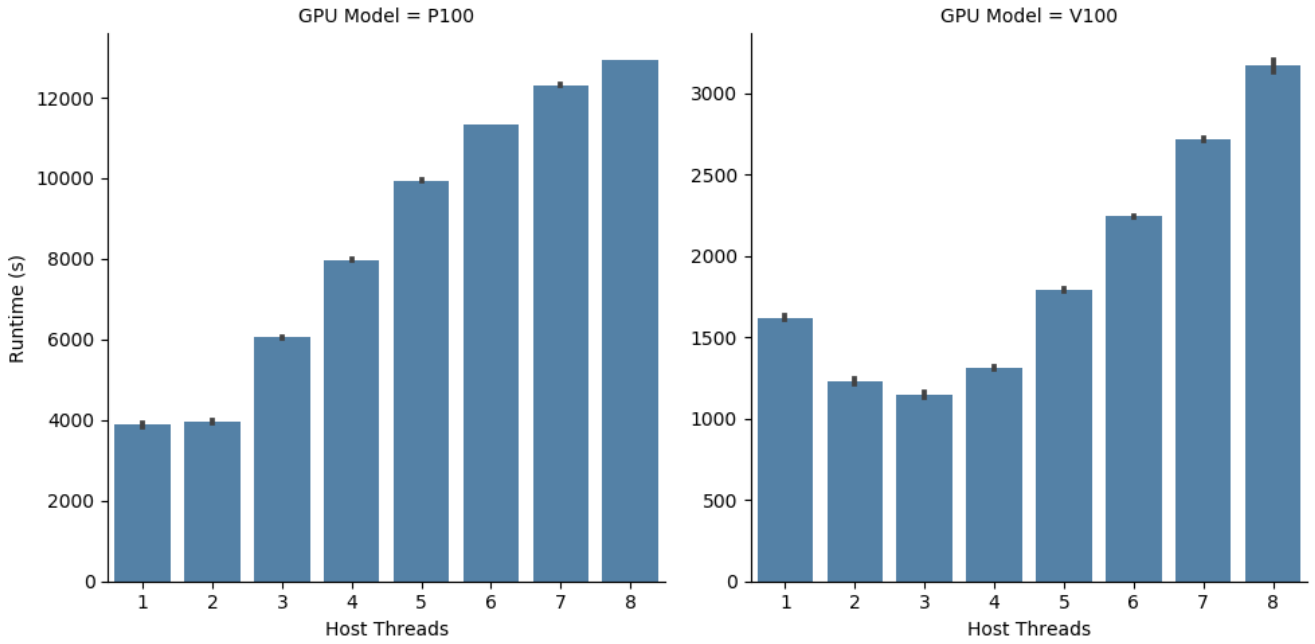

FIGURE 5. Effect of host threads on single-GPU performance. Each result is the average of five independent trials, with vertical lines denoting standard error.

four different implementations – *naive*, *memory-coalescing*, *workload-balancing*, and *coalescing-balancing* – using the average runtime of the GMM kernel as the performance metric. Due to the experimental nature of these optimizations, we used the *naive* implementation for all other benchmarks in this paper.

Lastly, we measured the speedup of KINCv3 GPU over KINCv3 CPU, as well as the speedup of KINCv3 CPU over KINCv1, for $\{1, 2, 4, 8\}$ workers. Note that the number of *workers* is one less than the number of *processes*, as the first process is solely responsible for distributing work and collecting results, with the exception of the single-CPU/GPU case, in which the process acts as both master and worker. Thus there is little difference between using one process versus using two processes, and in fact the difference in runtime between these two cases is negligible. For these experiments we used the default values given in Table 1 except for the number of host threads, for which we used the optimal value determined in the first set of experiments (2 threads for P100, 4 threads for V100).

VII. RESULTS AND DISCUSSION

A. MULTI-GPU EXECUTION PARAMETERS

Figure 6 shows runtimes for various work block sizes. Since a worker thread processes one work block at a time, the work block size is effectively the maximum number of gene pairs that a worker thread can process in parallel. The work block size has no effect on performance so long as it is greater than or equal to the global work size. Otherwise the global work size is effectively capped by the work block size, which can cause the GPU to be underutilized.

Figure 5 shows the effect of subscribing the GPU with multiple host threads. Since the default global work size may not scale to the capacity of newer GPU models, it can be supplemented by running multiple host threads on the same GPU to increase utilization. These results demonstrate that this approach does work in practice, as the optimal number of threads for the V100 is higher than for the P100, but there is a penalty in using too many threads, most likely from the overhead of managing more threads. It should also be noted that using multiple threads allows for kernel executions in one thread to be overlapped with memory transfers in another thread and *vice versa*, however this benefit is negligible since memory transfers make up a very small portion of GPU activity.

An alternative strategy to using more host threads is to increase the global work size to match the capacity of the GPU. We expected that this approach would perform as well or better than using multiple host threads, but the default global work size of 4096 resulted in the best performance out of all other values tried (Figure 7). Furthermore, we found from the GPU profiling results that the decrease in performance was caused by a disproportionate increase in the runtime of the GMM kernel, which we suspect was caused by increased global memory latency.

Figure 8 shows runtimes for various local work sizes. Increasing the local work size led to an increase in runtime in every case and for both P100 and V100. We attribute this trend to two primary factors: global memory bandwidth and work imbalance. Because the GMM kernel uses global memory very heavily, increasing the local work size also increases the amount of global memory traffic which hurts

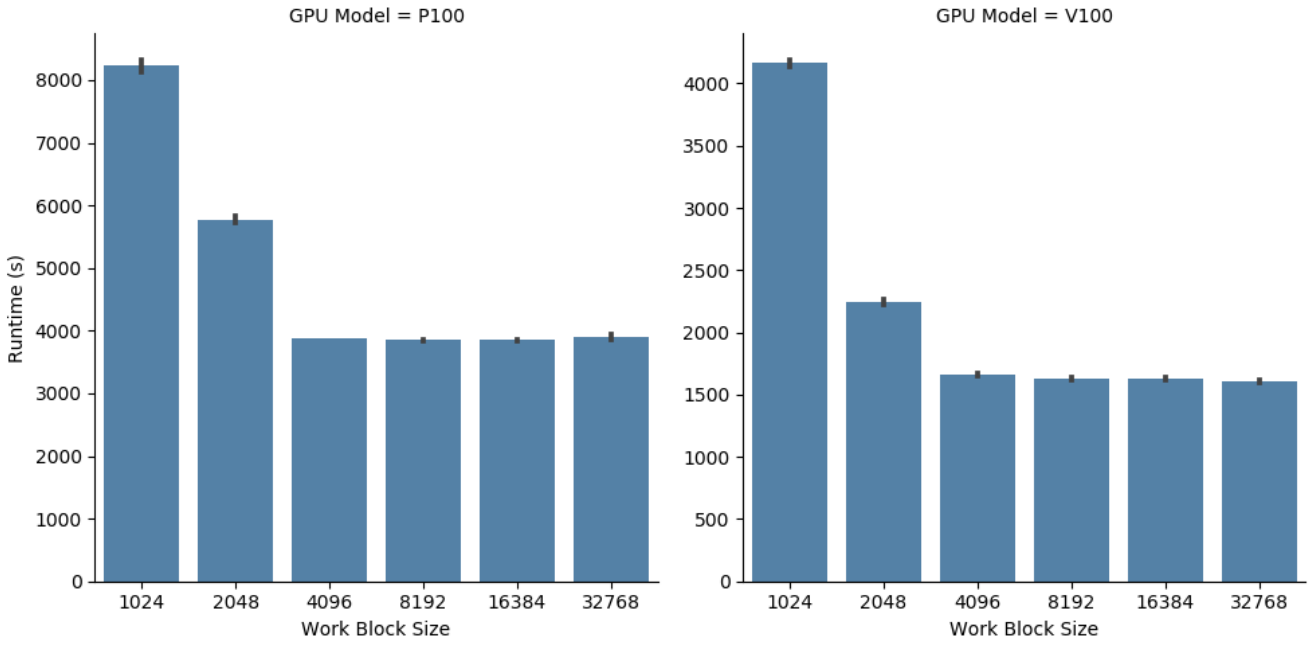

FIGURE 6. Effect of work block size on single-GPU performance. Each result is the average of five independent trials, with vertical lines denoting standard error.

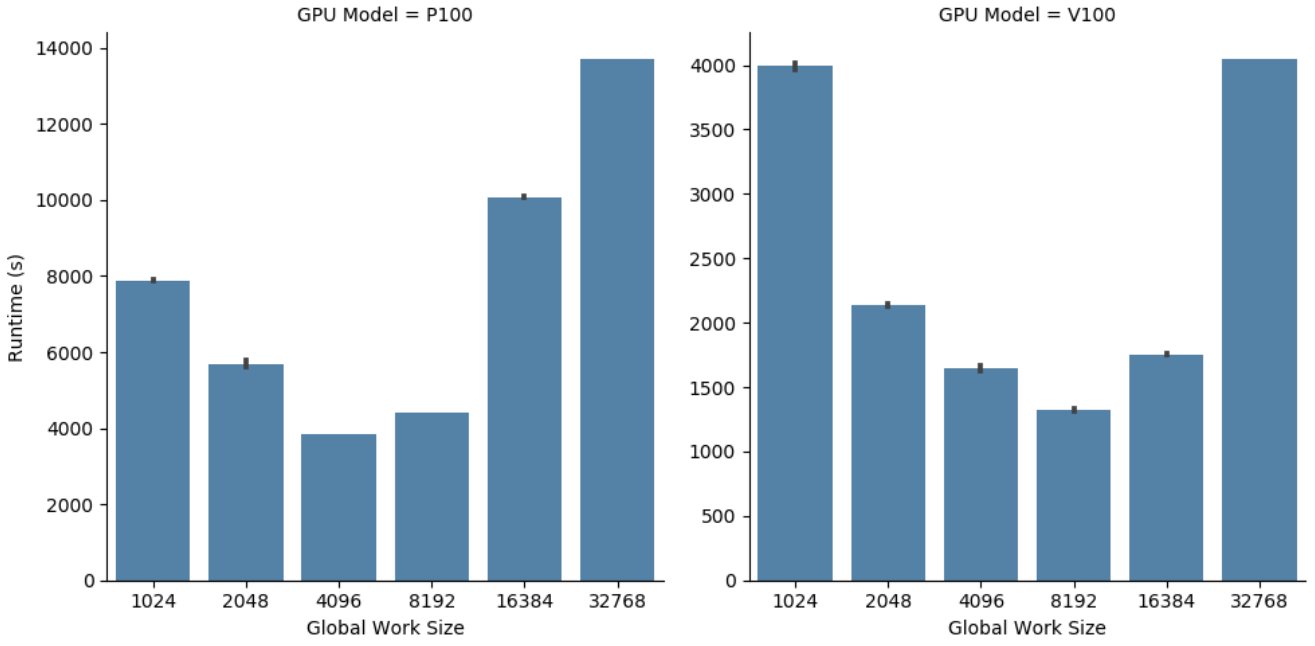

FIGURE 7. Effect of global work size on single-GPU performance. Each result is the average of five independent trials, with vertical lines denoting standard error.

overall performance. Additionally, due to the variation of sample sizes across gene pairs, increasing the local work size also increases the degree of work imbalance within a thread block. Within a CUDA warp, work imbalance causes control divergence as threads with fewer samples must wait for threads with more samples during loops. Across warps in a thread block, work imbalance simply causes higher latency as the total execution time of the thread block is determined by the warp which takes the longest. Therefore, it is better to use only a small percentage of compute bandwidth (low occupancy) so that global memory is not overwhelmed and the degree of work imbalance is minimized.

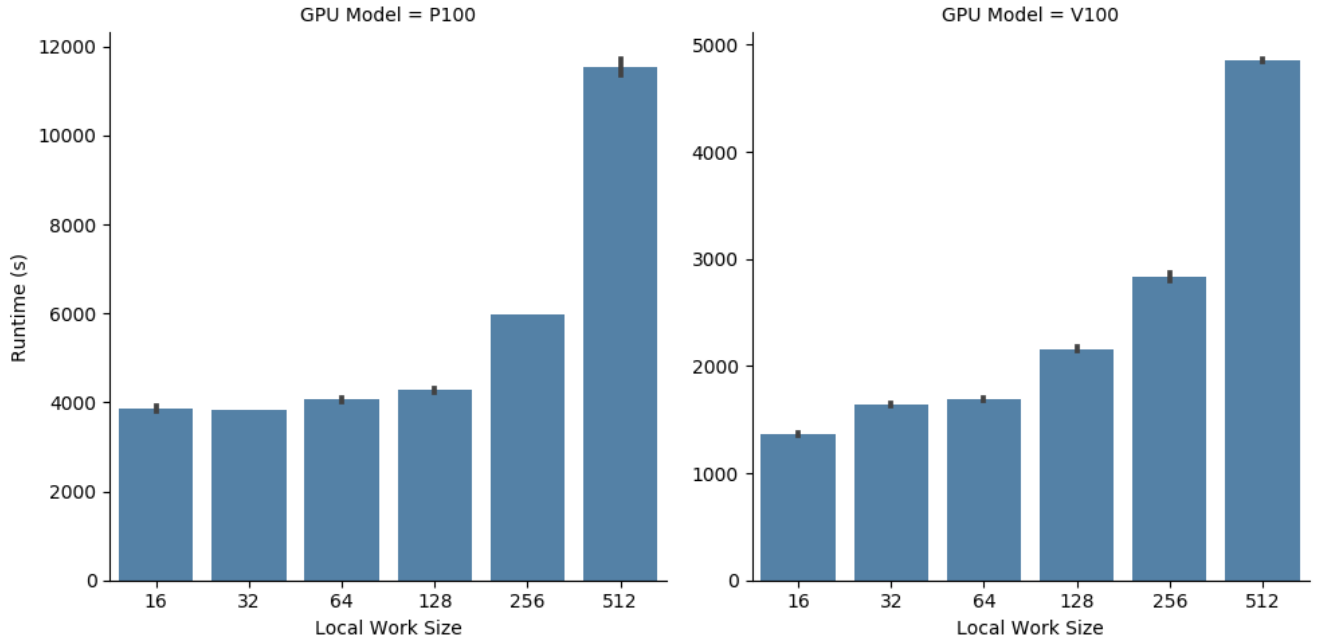

FIGURE 8. Effect of local work size on single-GPU performance. Each result is the average of five independent trials, with vertical lines denoting standard error.

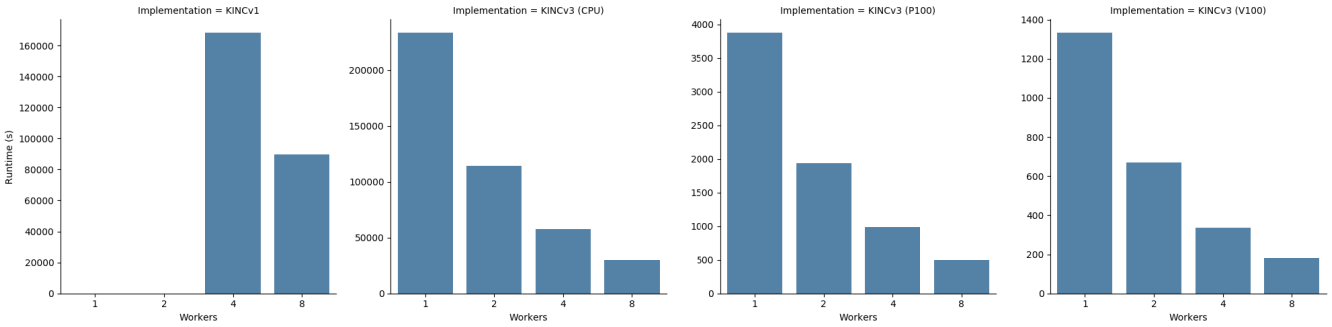

FIGURE 9. Performance benchmark results for KINCv1, KINCv3 CPU, and KINCv3 GPU implementations. Each result is the average of a single trial due to computational constraints. Note that the y-axis scale is different for each subplot in order to emphasize the scalability of KINCv3.

B. PERFORMANCE OPTIMIZATIONS

We measured the impact of each performance optimization on GMM kernel runtime, the results of which are shown in Figure 10. The results varied significantly depending on which GPU model was used. With older GPU models such as the Tesla K20 and K40, the *memory-coalescing* optimization reduced kernel runtime by half, but with newer models such as the Tesla P100 and V100, the same optimization has nearly the opposite effect. We attribute this difference to the advancements made between the Kepler and Pascal architectures. In particular, Pascal is the first GPU architecture with high-bandwidth memory (HBM2) DRAM [28]. Similarly, the *workload-balancing* optimization provided a small but noticeable improvement on the K20 and P100, and no improvement on the V100. We attribute the difference to advancements in the Volta architecture, specifically independent thread scheduling [29]. The hardware advancements in the most recent Tesla GPUs aim to increase memory bandwidth and thread convergence as they are some of the leading GPU performance bottlenecks, which could make hand-crafted optimizations such as ours unnecessary and even detrimental in some cases. Therefore, we conclude that these optimizations are only beneficial on older NVIDIA GPUs based on architectures prior to Pascal.

It should be noted that the two performance bottlenecks which we identified in the naive GMM kernel – high global memory usage and workload imbalance – are largely due to characteristics of gene expression matrices. RNA-Seq datasets, as well as other types of datasets in bioinformatics, are generally very large and diverse, with many sources of variation across samples. Therefore, while these datasets can generally be processed in a parallel manner, any algorithm which consumes them will be highly data-intensive and will experience some form of work imbalance due to the diversity of samples. In other words, we suspect that the performance bottlenecks identified in our multi-GPU implementation of

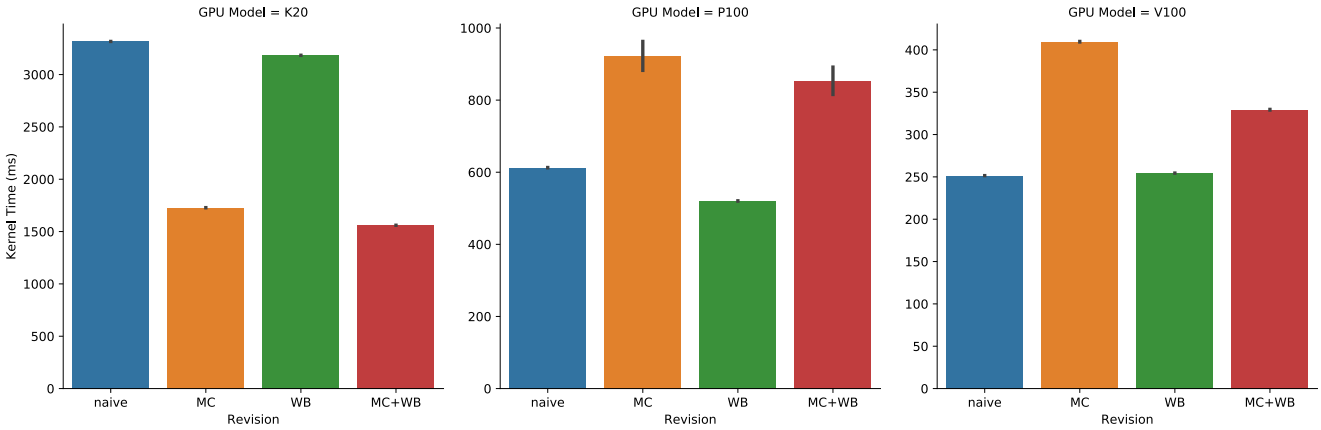

FIGURE 10. Effect of each performance optimization on kernel runtime. Each result is the average of three independent trials, with vertical lines denoting standard error. The measurement for each trial is itself the average of all GMM kernel calls required to process the yeast dataset. Note that the y-axis scale is different for each subplot in order to emphasize the performance differences between each revision.

TABLE 2. Performance benchmark results for KINCv1, KINCv3 CPU, and KINCv3 GPU implementations. Numbers in boldface denote the speedup values which were used to determine the speedup of KINCv3 over KINCv1.

	Runtime (s)				Speedup		
Workers	v1	v3 (CPU)	v3 (P100)	v3 (V100)	v3 (CPU)	v3 (P100)	v3 (V100)
1	-	233690	3882	1334	-	60.198	175.180
2	-	114545	1938	670	-	59.105	170.963
4	168389	57512	986	336	2.928	58.329	171.167
8	89740	30229	499	181	**2.969**	**60.579**	**167.011**

GCN construction are common to other bioinformatics applications for RNA-Seq datasets, and that any such optimizations which can alleviate these bottlenecks will be generally beneficial to bioinformatics workloads.

C. GPU SPEEDUP

Table 2 shows the speedup of the GPU implementation over the CPU implementation. KINCv3 GPU achieves a speedup of 38x on P100 GPUs and 126x on V100 GPUs. We also performed the same benchmark on KINCv1 to demonstrate the improvements to the CPU implementation from KINCv1 to KINCv3, which yielded a 3x speedup (results for some KINCv1 tests could not be obtained because the tests did not complete within the maximum walltime of 72 hours for jobs on our cluster). This speedup is attributed to a number of improvements, including: using a GMM implementation optimized for pairwise data, saving output data only for correlations with an absolute value of at least 0.5, and the dynamic work assignment provided by ACE. Thus the overall speedup of KINCv3 GPU over KINCv1 is 180x for the P100 and 496x for the V100.

The GPU implementation is also scalable for the range of processes that we tested, with a parallel efficiency of 94% for 8 V100s. The yeast dataset, although sizeable in its own right, is a relatively small gene expression dataset, containing fewer genes than multi-cellular organisms, yet KINCv3 is able to scale such that it can construct the similarity matrix for yeast in 3 minutes using 8 V100s.

More importantly, these speedup results demonstrate that KINCv3 can do an amount of work on a small GPU cluster that with KINCv1 would require a very large CPU cluster. Previously, large KINC jobs such as in [4] had to be run on a large high-throughput system such as the Open Science Grid (OSG) because they could not be completed on a local cluster due to resource and walltime constraints. However, jobs on OSG experience significant runtime overhead due to queuing times and re-running jobs that are preempted [30]. By accelerating KINC such that the same experiment can be performed on a much smaller scale, researchers can construct larger networks with local resources.

D. RUNNING KINC AT SCALE

Our experimental results demonstrated the speedup that KINCv3 provides for a relatively small dataset (7,050 genes, 188 samples) with a small set of resources (1-4 nodes). However, the long-term goal is to construct networks for very large datasets ($\geq$50,000 genes, $\geq$10,000 samples) with a large set of resources ($\geq$100 GPUs or $\geq$10,000 CPUs). Initial experiments have shown that while KINC remains computationally efficient at larger scales, its memory and storage requirements can expand rapidly in some cases. Therefore, it will be necessary to also profile and optimize the memory and disk usage of KINC at larger scales.

Additionally, KINC has two modes of parallel execution via ACE: a *high-performance (capability)* mode based on MPI, and a *high-throughput (capacity)* mode based on chunking and merging. These modes are independent of CPU/GPU execution and can be used interchangeably. The experiments in this paper used the high-performance mode, however at larger scales it is unclear whether one mode will give a lower time-to-solution than the other. The high-performance mode, on the one hand, is likely to

have better load balance due to the dynamic work assignment. On the other hand, the high-throughput mode is more flexible in the context of a shared HPC system because it can proceed gradually as cluster resources become available, whereas the high-performance mode must wait for all of the required resources to become available at once. Therefore it will also be important to compare the time-to-solution of these two modes by accounting for queue times in a large-scale shared HPC system.

VIII. CONCLUSION

We have presented a new implementation of KINC for constructing large multi-modal gene co-expression networks, which introduces both optimizations to the CPU code and GPU acceleration. After optimizing the execution parameters for KINCv3, we obtained up to 500x speedup from KINCv1 to KINCv3. Additionally, we identified key performance bottlenecks in the pairwise GMM kernel as well as several potential solutions, which we hope will improve the speedup even further. KINCv3 also shows good initial scalability, and we intend to conduct much larger experiments soon using both HPC and cloud environments. KINCv3 is open source software available at https://github.com/SystemsGenetics/KINC.

ACKNOWLEDGMENT

Benjamin T. Shealy and Josh J. R. Burns contributed equally to this work.

REFERENCES

[1] (2019). *SRA Overview*. [Online]. Available: https://trace.ncbi.nlm.nih.gov/Traces/sra/
[2] L. Song, P. Langfelder, and S. Horvath, "Comparison of co-expression measures: Mutual information, correlation, and model based indices," *BMC Bioinf.*, vol. 13, no. 1, 2012, Art. no. 328.
[3] (2019). *Systemsgenetics/Kinc*. [Online]. Available: https://github.com/SystemsGenetics/KINC
[4] S. P. Ficklin, L. J. Dunwoodie, W. L. Poehlman, C. Watson, K. E. Roche, and F. A. Feltus, "Discovering condition-specific gene co-expression patterns using Gaussian mixture models: A cancer case study," *Sci. Rep.*, vol. 7, no. 1, 2017, Art. no. 8617.
[5] W. L. Poehlman, J. J. Hsieh, and F. A. Feltus, "Linking binary gene relationships to drivers of renal cell carcinoma reveals convergent function in alternate tumor progression paths," *Sci. Rep.*, vol. 9, no. 1, 2019, Art. no. 2899.
[6] L. A. Honaas, H. L. Hargarten, S. P. Ficklin, J. A. Hadish, E. Wafula, C. W. de Pamphilis, J. P. Mattheis, and D. R. Rudell, "Co-expression networks provide insights into molecular mechanisms of postharvest temperature modulation of apple fruit to reduce superficial scald," *Postharvest Biol. Technol.*, vol. 149, pp. 27–41, Mar. 2019.
[7] O. Wilkins, C. Hafemeister, A. Plessis, M.-M. Holloway-Phillips, G. M. Pham, A. B. Nicotra, G. B. Gregorio, S. K. Jagadish, E. M. Septiningsih, and R. Bonneau, "EGRINs (environmental gene regulatory influence networks) in rice that function in the response to water deficit, high temperature, and agricultural environments," *Plant Cell*, vol. 28, no. 10, pp. 2365–2384, 2016.
[8] M. Aitkin and D. B. Rubin, "Estimation and hypothesis testing in finite mixture models," *J. Roy. Stat. Soc., B Methodol.*, vol. 47, no. 1, pp. 67–75, 1985.
[9] C. Fraley and A. E. Raftery, "How many clusters? Which clustering method? Answers via model-based cluster analysis," *Comput. J.*, vol. 41, no. 8, pp. 578–588, 1998.
[10] C. Biernacki, G. Celeux, and G. Govaert, "Assessing a mixture model for clustering with the integrated completed likelihood," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 7, pp. 719–725, Jul. 2000.
[11] P. Langfelder and S. Horvath, "WGCNA: An R package for weighted correlation network analysis," *BMC Bioinf.*, vol. 9, no. 1, p. 559, Dec. 2008.
[12] J. Petereit, S. Smith, F. C. Harris, Jr., and K. A. Schlauch, "petal: Co-expression network modelling in R," *BMC Syst. Biol.*, vol. 10, no. 2, p. 51, 2016.
[13] M. Liang, F. Zhang, G. Jin, and J. Zhu, "FastGCN: A GPU accelerated tool for fast gene co-expression networks," *PLoS ONE*, vol. 10, no. 1, 2015, Art. no. e0116776.
[14] S. M. Gibson, S. P. Ficklin, S. Isaacson, F. Luo, F. A. Feltus, and M. C. Smith, "Massive-scale gene co-expression network construction and robustness testing using random matrix theory," *PLoS ONE*, vol. 8, no. 2, 2013, Art. no. e55871.
[15] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *Proc. ICPR*, Aug. 2004, pp. 28–31.
[16] D. A. Reynolds and D. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Trans. Speech Audio Process.*, vol. 3, no. 1, pp. 72–83, Jan. 1995.
[17] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted Gaussian mixture models," *Digit. Signal Process.*, vol. 10, nos. 1–3, pp. 19–41, 2000.
[18] Y. Tarabalka, T. V. Haavardsholm, I. Kåsen, and T. Skauli, "Real-time anomaly detection in hyperspectral images using multivariate normal mixture models and GPU processing," *J. Real-Time Image Process.*, vol. 4, no. 3, pp. 287–300, 2009.
[19] V. Pham, P. Vo, L. H. Bac, and V. T. Hung, "GPU implementation of extended Gaussian mixture model for background subtraction," in *Proc. IEEE RIVF Int. Conf. Comput. Commun. Technol., Res., Innov., Vis. Future (RIVF)*, Nov. 2010, pp. 1–4.
[20] P. EhKan, T. Allen, and S. F. Quigley, "FPGA implementation for GMM-based speaker identification," *Int. J. Reconfigurable Comput.*, vol. 2011, Jan. 2011, Art. no. 3.
[21] M. Genovese and E. Napoli, "ASIC and FPGA implementation of the Gaussian mixture model algorithm for real-time segmentation of high definition video," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 537–547, Mar. 2014.
[22] W. G. Lewellen. (2019). *lewellen/Gaussianmixturemodel*. [Online]. Available: https://github.com/lewellen/gaussianMixtureModel
[23] C. Guo, H. Fu, and W. Luk, "A fully-pipelined expectation-maximization engine for Gaussian mixture models," in *Proc. Int. Conf. Field-Programmable Technol.*, Dec. 2012, pp. 182–189.
[24] C. Biernacki, G. Celeux, G. Govaert, and F. Langrognet, "Model-based cluster and discriminant analysis with the mixmod software," *Comput. Statist. Data Anal.*, vol. 51, no. 2, pp. 587–600, 2006.
[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[26] S. Ficklin and B. Shealy. (Jan. 2019). *SystemsGenetics/ACE: Version 3.0.2*. [Online]. Available: https://doi.org/10.5281/zenodo.2539525
[27] T. Barrett, D. B. Troup, S. E. Wilhite, P. Ledoux, C. Evangelista, I. F. Kim, M. Tomashevsky, K. A. Marshall, K. H. Phillippy, and P. M. Sherman, "NCBI GEO: Archive for functional genomics data sets—10 years on," *Nucleic Acids Res.*, vol. 39, pp. D1005–D1010, Nov. 2010.
[28] *P100, Most Advanced Data center Accelerator Ever Built Featuring Pascal GP100, Worlds Fastest GPU*, Nvidia, Santa Clara, CA, USA, 2016.
[29] *V100 GPU Architecture*, Nvidia, Santa Clara, CA, USA, 2017.
[30] W. L. Poehlman, M. Rynge, D. Balamurugan, N. Mills, and F. A. Feltus, "OSG-KINC: High-throughput gene co-expression network construction using the open science grid," in *Proc. IEEE Int. Conf. Bioinf. Biomed. (BIBM)*, Nov. 2017, pp. 1827–1831.

BENJAMIN T. SHEALY received the B.S. degree in computer engineering from Clemson University, in 2017, where he is currently pursuing the Ph.D. degree in computer engineering. He has several years of experience developing high-performance applications for computer vision, bioinformatics, and computational materials science. His current research focuses on creating big data pipelines that utilize emerging computing technologies such as GPUs, cluster computing, and cloud computing. He is also interested in applying machine learning to big data pipelines, both to improve their execution at scale and to assist in the analysis of their results. His work is very interdisciplinary in nature and he collaborates regularly with biologists, computational scientists, and software engineers in order to facilitate his work.

JOSH J. R. BURNS received the B.S. degree in electrical engineering from the University of Idaho, in 2013. He is currently working as a Software Engineer with Washington State University. He is a self-taught programmer with almost two decades of experience. His primary areas of expertise are C++ and the Qt framework. His primary interest in the field of programming has always been making code as human-readable as possible. He has worked with OpenCL in developing GPU-accelerated computational algorithms at Washington State University.

MELISSA C. SMITH received the B.S. and M.S. degrees in electrical engineering from Florida State University, 1993 and 1994, respectively, and the Ph.D. degree in electrical engineering from the University of Tennessee, in 2003. She is currently an Associate Professor of electrical and computer engineering with Clemson University. She has over 25 years of experience developing and implementing scientific workloads and machine learning applications across multiple domains, including 12 years as a Research Associate with Oak Ridge National Laboratory. Her current research focuses on the performance analysis and optimization of emerging heterogeneous computing architectures (GPGPU- and FPGA-based systems) for various application domains, including machine learning, high-performance or real-time embedded applications, and medical and image processing. Her lab collaborates with researchers in other fields to develop new approaches to the application/architecture interface providing interdisciplinary solutions that enable new scientific advancements and/or capabilities.

F. ALEX FELTUS received the B.S. degree in biochemistry from Auburn University, in 1992. He has served in the Peace Corps in the Fiji Islands for two years, and then completed advanced training in biomedical sciences at Vanderbilt and Emory. He has performed research in machine learning, bioinformatics, cyberinfrastructure, high-performance computing, network biology, tumor biology, agrigenomics, genome assembly, systems genetics, paleogenomics, and bioenergy feedstock genetics. He is currently a Professor with the Department of Genetics and Biochemistry, Clemson University, and the CEO of Allele Systems LLC. He has published numerous scientific articles in peer-reviewed journals, released open-source software, and taught undergrad and Ph.D. students in bioinformatics, biochemistry, and genetics. He is funded by multiple NSF grants and is engaged in tethering together extremely smart people from diverse technical backgrounds to propel genomics research from the Excel-scale into the Exascale.

STEPHEN P. FICKLIN received the B.S. degree in computer science from Brigham Young University, in 2000, the M.S. degree in computer science from Clemson University, in 2003, and the Ph.D. degree in plant and environmental sciences from Clemson University, in 2013. He also served professionally as a UNIX Systems Administrator, a Bioinformaticist, and an Assistant Director of bioinformatics with the Clemson's Genomics Institute for eight years. He was a Researcher and held a postdoctoral position with Washington State University, for five years. He is currently an Assistant Professor with a research and teaching appointment at the Department of Horticulture, Washington State University, with an emphasis on bioinformatics and computational systems biology. His focus is the development of computational tools and infrastructure that improve our understanding of the molecular mechanisms that underlie complex traits of agricultural crops. Such traits are often important economically, benefit human health, and improve food security, management, and resource utilization. He teaches courses in data science and systems biology to graduate-students across life sciences.