1 GENOME ANALYSIS

Efficient Detection of Highly Mutated Regions with Mutations Overburdening Annotations Tool (MOAT)

Lucas Lochovsky[1,2†], Jing Zhang[1,2†] and Mark Gerstein[1,2,3*]

[1]Program in Computational Biology and Bioinformatics, Yale University, New Haven, Connecticut 06520, USA
[2]Department of Molecular Biophysics and Biochemistry, Yale University, New Haven, Connecticut 06520, USA
[3]Department of Computer Science, Yale University, New Haven, Connecticut 06520, USA

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Identifying genomic regions with a higher-than-expected mutation burden (i.e. overburdened regions) has a number of useful applications. In the context of somatic cancer variants, overburdened regions may be highly associated with cancer progression. In the germline, accumulation of rare variants could be an indicator of positive selection. Mutation burdens could also be applied in the context of *de novo* mutations to identify precursor variants for genetic disease. Here, we release a new GPU-based computational tool, called MOAT (the Mutations Overburdening Annotations Tool), to perform mutation burden analysis with ultrafast speed. MOAT takes no assumption about the mutation process except that the background mutation rate (BMR) changes smoothly with other genomic features. This nonparametric scheme randomly permutes the variants (or target regions) on a relatively large scale where the BMR is assumed to be constant to provide robust burden analysis in various scenarios. Furthermore, it also incorporates a somatic variant simulator called MOATsim, which randomly permutes the input variants with effective covariate control. In conclusion, MOAT will be useful for a broad range of analyses that would benefit from a methods evaluation on variant permutation data.

Availability: MOAT is available at moat.gersteinlab.org

2 INTRODUCTION

A common analysis strategy in high throughput sequencing is to look for genomic elements with a high accumulation of variants across a large cohort of patients. However, it is well known that the background mutation rate (BMR) is highly heterogeneous across the whole genome due to numerous external features. For example, replication timing and chromatin structure usually affects the BMR at a scale of up to one megabase (Lochovsky, et al., 2015). Such effects may change in a dynamic way across the genome, and is usually difficult to model. Hence, our Mutations Overburdening Annotations Tool (MOAT) relies on no assumption except that the BMR changes slowly across the genome and approximately remains the same within a local context. Therefore, this nonparametric scheme provides robust mutation burden significance results for any element through permutations.

MOAT offers two methods for determining elevated mutation burdens, the annotation-centric algorithm (MOAT-a) and the variant-centric algorithm (MOAT-v), both of which involve a comparison of each annotation's mutation accumulation to that of the surrounding genome. We also offer MOATsim, a variant distribution simulator based on MOAT-v's source code, which reflects the distribution of whole genome covariates that influence the BMR. In the following sections, we will describe the implementation of MOAT for parallel computer systems, which enables highly efficient data size scalability. We also evaluate MOAT's ability to recall known noncoding cancer drivers on a collection of several hundred cancer whole genomes' variant data.

3 METHODS

A number of covariates jointly affect the BMR in a complicated and dynamic manner, making variant burden analysis very challenging (Lochovsky, et al., 2015). However, the length of target region usually varies from hundreds of bases (such as transcription start sites) to thousands of bases (such as enhancers), while external features, such as replication timing, work at up to megabase resolution. Therefore, MOAT circumvents the necessity for such parametric models by explicitly permuting the variants or annotations within a region that has essentially constant levels of all the covariates. It assumes constant covariate values over a fairly large interval—with a typical size of ~100kb—that is appreciably larger that the annotations we wish to assess for elevated mutation burdens.

One important issue with these permutation algorithms is that their running times do not scale well to noncoding annotation sets. A typical burden analysis usually consists of more than 10^5 annotations (such as all transcription start sites of GENCODE annotation), with 1000 permutations per region. It can easily expand to billions of permutation and intersection calculations, making the overall operation very compute intense. Our analyses indicate that an unoptimized run that involves ~3 million annotations with 1000 permutations each would take about 14 hours to complete.

*To whom correspondence should be addressed.
†The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors.

Style Definition: Normal

Deleted: such

Deleted: A major challenge of working with somatic cancer variants is the need to identify those variants responsible for disruptions that drive cancer progression out of the thousands that arise due to background mutation processes. One approach is to scan the genome for elements with a high frequency of intersecting somatic variants, or elevated mutation burdens. The detection of significant mutation burdens is also useful in germline variant analysis, as rare variant burdens may indicate increased risk of genetic disease.. Here, we introduce the Mutations Overburdening Annotations Tool (MOAT), a new computational tool designed to identify regions with a high mutation burden relative to the surrounding genome in a non-parametric way. MOAT is useful for prioritizing annotations to study in downstream analyses, as high mutation burden annotations are most likely to be driver elements in genetic disease. We release an implementation that offers users two forms of mutation burden analysis through empirical permutations, as well as serial and parallel versions of each form. We also demonstrate MOAT's capability for finding known noncoding drivers in cancer variant data.

Deleted: of

Deleted: This scalability is important for guaranteeing a reasonable running time given the high computational intensity of the permutation step.

Deleted: High

Formatted: Indent: First line: 0"

Deleted: of genetic disease cohorts has enabled the identification of the molecular causes of these illnesses. This data can be utilized to find the somatic single nucleotide variants (SNVs) in each patient. However, due to the relatively high number of neutral variants in such patients' genomes, it is not immediately apparent which variants are directly connected to the disease phenotype. A common strategy for addressing this issue

Deleted: . By modeling the factors that influence the stochastic mutation rate, the elements that are more mutated than expected under the background model can be determined. ... [1]

Deleted: , such as

Deleted: This effect

Deleted: background mutation rate

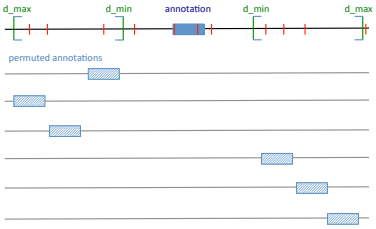


Figure 1 For each input annotation, MOAT-a finds the number of intersecting *vfile* variants (red). The annotation's coordinates are then shuffled to a new location within the local genome context bounded by user-defined parameters *d_min* and *d_max*, producing *n* permutations (blue). Each permutation's intersecting variant count is computed.

Here we have addressed this compute intensity by taking advantage of very large-scale parallelization available on modern GPU technology. The key realization behind this design choice is the fact that MOAT's most intense operations consist of a huge number of mathematical calculations with no data dependencies between each calculation, which plays perfectly into the strengths of general purpose GPU pipelines. In the following sections, we demonstrate MOAT's parallelization schemes for both the variant-based and annotation-based permutation algorithms.

3.1 MOAT-a: Annotation-centric Permutation

MOAT takes two input files: the annotation file (*afile*) and the variant file (*vfile*). The parallel version of MOAT's annotation-centric permutation algorithm, MOAT-a, is a C++ program that uses NVIDIA's CUDA language to instantiate parallel graphics processing unit (GPU) threads, and divides the computational workload across these threads. MOAT-a's steps are illustrated in Fig. 1. MOAT-a iterates through the annotations, computing the intersecting variant count per annotation. It then defines a genomic block with user-defined boundaries centered on the current input annotation, and randomly moves the annotation within this block. MOAT-a will find the variant counts from the *vfile* that intersect each of the random bins, which are compared to the input annotation's variant count. The input annotation's p-value is defined as the fraction of bins with a variant count equal to or greater than the input annotation's variant count.

In the GPU version, the input variants and annotations are copied to the video RAM (VRAM), making the data accessible to the GPU's threads. The annotations are divided among the threads, and each thread iterates through its subset of annotations, finding the observed variant count for each annotation, computing the permuted locations, and calculating the permuted variant counts. The permuted annotation locations are temporarily stored to calculate the variant counts and derive the annotation's p-value, after which the memory is freed for the next annotation. Only the p-values are persistently stored, which controls MOAT-a's usage of VRAM—a far more limited resource compared to main memory. The p-values are then copied back to main memory for generating the program's output.

3.2 MOAT-v: Variant-centric Permutation

MOAT-v's variant-centric permutation algorithm creates permuted datasets by assigning new coordinates to each variant within a local genome region to account for the covariate effects from known genomic features (Fig. 2a). MOAT-v preserves the trinucleotide context of the original variant when choosing a new variant location. In other words, the new variant must have the same nucleotide identity as the original variant, and the two neighbors of the new variant must also have the same nucleotide identity as the original variant's neighbors. For example, if MOAT-v is given an input variant that has a reference base G, and is surrounded by a T and a C (i.e. the variant's trinucleotide context is TGC), then MOAT-v gathers up every position in the same bin where TGC occurs in the reference, and selects one of these with uniform probability (Fig. 2b). This constraint reflects the differential mutation probabilities of different trinucleotides (i.e. due to biochemical differences, some trinucleotides are more likely to be mutated than others), and ensures that the permuted variant set does not change the expected distribution of mutated trinucleotides. Hence, MOAT-v preserves the mutational signature of the input variants.

MOAT-v takes a *vfile* and an *afile* as inputs, and generates a permuted dataset by subdividing the genome into blocks of a user-defined size (excluding mappability blacklist regions), and assigning each block's variants new positions within the same block, preserving trinucleotide context in the process. This process continues until *n* permutations have been generated. At this point, MOAT-v will calculate *n* intersecting permuted variant counts for each of the input annotations. A p-value for each annotation is determined based on the fraction of the *n* intersecting permuted variant counts that are equal to or greater than the count derived from the original *vfile* variants.

3.3 MOATsim: Simulated Somatic Variant Datasets

In addition to the main MOAT programs, we have developed a somatic variant simulator, MOATsim, that reflects the levels of whole genome covariates that directly influence the background mutation rate. It is based off the variant permutation step in MOAT-v, but also incorporates bigWig signal files in its determination of permuted variant locations. Before working with the actual variant data, MOATsim imports the covariate data, and

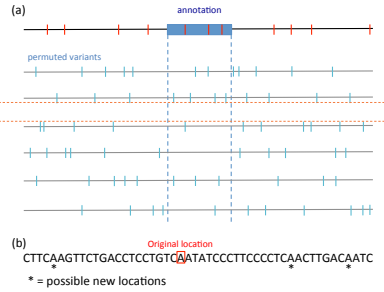


Figure 2 (a) In MOAT-v, the variant locations are permuted within the local genome context. The whole genome is divided into bins of a user-defined size, and variants are moved to new coordinates within the same bin, preserving the local mutation context. As with MOAT-a, *n* permutations are produced. (b) To reflect the influence of nucleotide identity on mutation likelihood, MOAT-v ensures that variants are moved to locations with the same trinucleotide context.

Deleted: *K.Takahashi*

Deleted: These covariates influence the whole genome background mutation rate, hence they must be taken into consideration when assessing an annotation's mutation burden relative to background mutation. For small enough regions, we assume these covariates are essentially constant, and we can perform variant permutations under the assumption of uniformity, with one key constraint. MOAT-v must preserve

Deleted: bins

Deleted: bin's

Deleted: bin

Deleted: <sp>

Deleted: intersecting variant

Deleted: MOAT takes two input files: the annotation file (*afile*) and the variant file (*vfile*).

Deleted:

Deleted: Initial prototypes of the parallel version of MOAT-v used the Nvidia CUDA framework, but the necessity of loading the reference genome sequence to preserve trinucleotide context in the permutation step resulted in prohibitive memory requirements with respect to the available GPU video RAM. As a result, MOAT-v was instead written to parallelize its workflow across multi-core CPUs using the OpenMPI framework (Gabriel, et al., 2004). Under this arrangement, a single CPU core is designated to run the "master" process, and is responsible for dividing up the overall work and distributing it to the "worker" processes, which run on the remaining cores.

Deleted: (Nickolls, et al., 2008)

Deleted: an extended region

Deleted: a

Deleted: distance

Deleted: at

Deleted: extended region

Deleted:

Deleted: <sp><sp>A typical MOAT-a run involves an annotation count on the order of ~10^5 at a minimum, each of which are permuted 1000 times. Hence, the overall computation involves millions—or even billions—of permutation and intersection calculations, which take an inordinate amount of time to complete. However, this computation is very easily adaptable to parallel execution, so MOAT-a breaks up the overall computation into many separate, independent units of computation. ... [2]

Deleted: <sp>

evaluates covariate signals over a set of whole genome bins (user-defined size, mappability blacklist regions excluded). These bins are then clustered based on their covariate signal profile using k-means clustering. With this information, variants can be permuted not just within their local genome context, but across all bins with the same covariate signal profile (i.e. within the same cluster). This additional functionality enhances MOATsim's accurate reflection of the expected somatic variant distribution of a human genome.

4 RESULTS

4.1 MOAT-a

Table 1. Speed benchmark of MOAT-a with respect to the number of input annotations and MOAT-v with respect to the number of parallel CPUs. MOAT-a's time trials involved generating 1000 permuted variant datasets, while MOAT-v's time trials involved generating one permuted variant dataset using ~8 million input variants, and 1,000,000-bp bins. For large datasets, the GPU algorithm substantially outperforms the CPU version.

MOAT-a		MOAT-v	
Annotations	GPU speedup	Parallel CPUs	Speedup
~14,000	1.01x	2	1.97x
~130,000	1.34x	4	3.50x
~3,000,000	6.26x	8	5.60x

We demonstrate the magnitude of the CUDA speedup by evaluating the running time of MOAT-a on datasets of various sizes. We took a dataset of pancancer whole genome variant calls that includes 507 cancer genomes of various types from (Alexandrov, et al., 2013), and 100 stomach cancer genomes from (Wang, et al., 2014), totaling ~8 million variants. We used 3 different annotation sets as input to demonstrate MOAT-a's scalability. We tested MOAT-a's running time on these 3 annotation sets with the number of random bins n = 1000 (Table 1, left half). It is clear that when scaling up to very large datasets, the CPU version's runtime increases considerably, while the GPU version's runtime rises very gradually.

Due to the relative lack of verified noncoding regulatory elements associated with cancer, it is difficult to assess the accuracy of MOAT-a's predictions. Nevertheless, we demonstrate MOAT-a's usefulness for finding elevated mutation burdens in genomic elements by identifying highly mutated GENCODE transcription start sites, promoters, and enhancers, using the aforementioned pancancer variant dataset. TERT, which has well-documented cancer-associated promoter mutations (Vinagre, et al., 2013), was found to have two TSSes with significant mutation burden (both had BH-corrected p-values of zero). Other well-known cancer-associated TSS sites, including TP53, LMO3, and AGAP5, also had significant mutation burdens (all had BH-corrected p-values of zero).

4.2 MOAT-v

Using the same set of cancer variants used in the MOAT-a tests, parallel MOAT-v's running time was evaluated across multiple CPU configurations to demonstrate the performance gains of the OpenMPI implementation. MOAT-v in OpenMPI is set up to run one master process on one of the available CPU cores, and use the rest for worker processes. Hence, the program must be run with 3 cores to get two cores to process the work simultaneously, 4 cores to get three cores to process the work simultaneously, etc. Table 1 (right half) represents the running time improvement relative to the number of workers added. This improvement scales close to linear with the number of workers, indicating that the load balancing between each CPU core is very evenly divided, enabling significant time savings when MOAT-v is run in parallel. MOATsim's running time exhibited similar characteristics (data not shown).

MOAT-v was used on the same variant and annotation sets used to demonstrate MOAT-a's usefulness for finding elevated cancer mutation burdens. MOAT-v produced comparable results—the same known cancer-associated TSSes flagged as significant in MOAT-a were also flagged in MOAT-v.

5 DISCUSSION

Identification of genomic elements with a high mutation burden is useful for narrowing down the exact site of functional disruption. We introduce Mutations Overburdening Annotations Tool (MOAT), a new software tool to facilitate such analyses. We demonstrate the usefulness of this tool for flagging putative noncoding cancer drivers, and provide CUDA- and OpenMPI-accelerated versions that dramatically increase the speed of mutation burden analysis. Given the demand for efficient, meaningful analysis of genome sequence data that is now being produced at a very high rate, we consider MOAT's provision of such analysis for genetic disease drivers quite timely.

Funding: This work was supported by the National Institutes of Health [5U41HG007000-04].

REFERENCES

Alexandrov, L.B., *et al*. Signatures of mutational processes in human cancer. *Nature* 2013;500(7463):415-421.

Lochovsky, L., *et al*. LARVA: an integrative framework for large-scale analysis of recurrent variants in noncoding annotations. *Nucleic acids research* 2015;43(17):8123-8134.

Vinagre, J., *et al*. Frequency of TERT promoter mutations in human cancers. *Nature communications* 2013;4:2185.

Wang, K., *et al*. Whole-genome sequencing and comprehensive molecular profiling identify new driver mutations in gastric cancer. *Nature genetics* 2014;46(6):573-582.

Deleted: Table 2. Speed benchmark of MOAT-v with respect to the number of CPU cores assigned worker processes. Each time trial involved using MOAT-v to generate one permuted variant dataset using ~8 million input variants, and 1,000,000-bp bins.
Running time ... [7]

Deleted: 2

Deleted: (CPU and GPU versions) …ith respect to t ... [3]

Deleted: After applying BH correction to all p-values, there were 1394 promoters, 451 TSSes, and 109 DRMs with significant mutation burdens. Hence, MOAT-v appears to be the more conservative algorithm.

Formatted: Font:Times

Formatted: Indent: First line: 0"

Deleted: Annotation set

Deleted: Number of annotations

Deleted: CPU version running time

Deleted: GPU version running time

Formatted Table

Deleted: Fold…PU speedup of GPU version ... [4]

Deleted: DRM

Deleted: 1hr23min

Deleted: 1hr22min

Deleted: TSS

Deleted: 1hr55min

Deleted: 1hr26min

Deleted: DHS

Deleted: 13hr46min

Deleted: 2hr12min

Deleted: , using both the CPU and GPU versions to calculate the output.… We took a dataset of pan- ... [5]

Deleted: Benjamini, Y. and Hochberg, Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*. 1995;57(1):289-300.

Nickolls, J., *et al*. Scalable parallel programming with CUDA. *Queue* 2008;6(2):40-53. ... [8]

Deleted: MOAT's…OAT-a's predictions. Neverthe ... [6]

Deleted: Nickolls, J., *et al*. Scalable parallel programming CUDA. *Queue* 2008;6(2):40-53. ... [9]

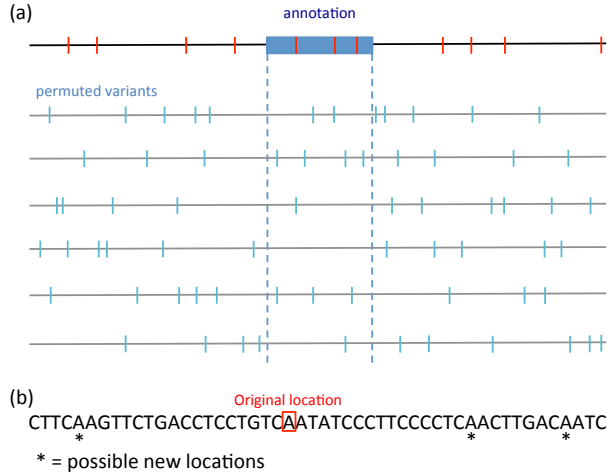
Deleted: Yip, K.Y., *et al*. Classification of human genomic regions based on experimentally determined binding sites of more than 100 transcription-related factors. *Genome biology* 2012;13(9):R48.

Page 1: [1] Deleted LL+JZ 8/23/16 5:45:00 PM

. By modeling the factors that influence the stochastic mutation rate, the elements that are more mutated than expected under the background model can be determined.
It is

Page 2: [2] Deleted LL+JZ 8/23/16 5:45:00 PM

Figure 2 (a) In MOAT-v, the variant locations are permuted within the local genome context. The whole genome is divided into bins of a user-defined size, and variants are moved to new coordinates within the same bin, preserving the local mutation context. As with MOAT-a, *n* permutations are produced. (b) To reflect the influence of nucleotide identity on mutation likelihood, MOAT-v ensures that variants are moved to locations with the same trinucleotide context.



A typical MOAT-a run involves an annotation count on the order of ~10^5 at a minimum, each of which are permuted 1000 times. Hence, the overall computation involves millions—or even billions—of permutation and intersection calculations, which take an inordinate amount of time to complete. However, this computation is very easily adaptable to parallel execution, so MOAT-a breaks up the overall computation into many separate, independent units of computation.

MOAT-a's speed can be further improved by taking advantage of the thousands of parallel stream processors available on modern graphics processing units (GPUs). GPUs are designed for efficiently processing 3D graphics calculations, which largely consist of numerous matrix operations with relatively low memory usage Due to the limited amount of video RAM (VRAM) available on GPUs, MOAT-a's GPU version was planned to use GPU acceleration only for the most compute intense step. This is the permutation step, where new annotation locations are determined in the local genome surrounding each annotation. The annotation coordinates are copied to VRAM, and one permutation per annotation is calculated in parallel. The coordinates for the permuted annotations are copied back to main memory for the fast intersection step. This permutation/intersection loop is performed n times (the user-defined total number of permutations), after which p-values are calculated using the observed variant counts and the permuted variant counts (Fig. 1).

Page 3: [3] Deleted LL+JZ 8/23/16 5:45:00 PM

(CPU and GPU versions)

Page 3: [3] Deleted LL+JZ 8/23/16 5:45:00 PM

(CPU and GPU versions)

Page 3: [3] Deleted LL+JZ 8/23/16 5:45:00 PM

(CPU and GPU versions)

Page 3: [3] Deleted LL+JZ 8/23/16 5:45:00 PM

(CPU and GPU versions)

Page 3: [3] Deleted LL+JZ 8/23/16 5:45:00 PM

(CPU and GPU versions)

Page 3: [3] Deleted LL+JZ 8/23/16 5:45:00 PM

(CPU and GPU versions)

Page 3: [3] Deleted LL+JZ 8/23/16 5:45:00 PM

(CPU and GPU versions)

Page 3: [4] Deleted LL+JZ 8/23/16 5:45:00 PM

Fold

Page 3: [4] Deleted LL+JZ 8/23/16 5:45:00 PM

Fold

Page 3: [5] Deleted LL+JZ 8/23/16 5:45:00 PM

, using both the CPU and GPU versions to calculate the output.

Page 3: [5] Deleted LL+JZ 8/23/16 5:45:00 PM

, using both the CPU and GPU versions to calculate the output.

Page 3: [5] Deleted LL+JZ 8/23/16 5:45:00 PM

, using both the CPU and GPU versions to calculate the output.

Page 3: [5] Deleted LL+JZ 8/23/16 5:45:00 PM

, using both the CPU and GPU versions to calculate the output.

Page 3: [5] Deleted LL+JZ 8/23/16 5:45:00 PM

, using both the CPU and GPU versions to calculate the output.

Page 3: [5] Deleted LL+JZ 8/23/16 5:45:00 PM

, using both the CPU and GPU versions to calculate the output.

Page 3: [5] Deleted LL+JZ 8/23/16 5:45:00 PM

, using both the CPU and GPU versions to calculate the output.

Page 3: [5] Deleted LL+JZ 8/23/16 5:45:00 PM

, using both the CPU and GPU versions to calculate the output.

Page 3: [6] Deleted LL+JZ 8/23/16 5:45:00 PM

MOAT's

Page 3: [6] Deleted LL+JZ 8/23/16 5:45:00 PM

MOAT's

Page 3: [6] Deleted LL+JZ 8/23/16 5:45:00 PM

MOAT's

Page 3: [6] Deleted LL+JZ 8/23/16 5:45:00 PM

MOAT's

Page 3: [7] Deleted LL+JZ 8/23/16 5:45:00 PM

Table 2. Speed benchmark of MOAT-v with respect to the number of CPU cores assigned worker processes. Each time trial involved using MOAT-v to generate one permuted variant dataset using ~8 million input variants, and 1,000,000-bp bins.

# of worker CPU cores	Running time	Fold speedup
1	3hr44min	1.00x
2	1hr54min	1.97x
4	1hr4min	3.50x
8	40min	5.60x

Page 3: [8] Deleted LL+JZ 8/23/16 5:45:00 PM

Benjamini, Y. and Hochberg, Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*. 1995;57(1):289-300.

Gabriel, E., *et al*. Open MPI: Goals, concept, and design of a next generation MPI implementation. *Springer* 2004:97-104.

Harrow, J., *et al*. GENCODE: the reference human genome annotation for The ENCODE Project. *Genome research* 2012;22(9):1760-1774.

Page 3: [9] Deleted LL+JZ 8/23/16 5:45:00 PM

Nickolls, J., *et al*. Scalable parallel programming with CUDA. *Queue* 2008;6(2):40-53.

Thurman, R.E., *et al*. The accessible chromatin landscape of the human genome. *Nature* 2012;489(7414):75-82.