Scaling Out a Combinatorial Algorithm for Discovering Carcinogenic Gene Combinations to Thousands of GPUs

Sajal Dash
National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN - 37830
Email: dashs@ornl.gov

Qais Al-Hajri
Computer and Electrical Engineering
Virginia Tech
Blacksburg, VA - 24060
Email: qaisfa96@vt.edu

Wu-chun Feng
Computer Science
Virginia Tech
Blacksburg, VA - 24060
Email: feng@cs.vt.edu

Harold R Garner
Biomedical Sciences
Edward Via College of Osteopathic Medicine
Spartanburg, SC - 29301
Email: sgarner@vt.vcom.edu

Ramu Anandakrishnan
Biomedical Sciences
Edward Via College of Osteopathic Medicine
Blacksburg, VA - 24060
Email: ramu@vt.edu

Abstract*—Cancer is a leading cause of death in the US, second only to heart disease. It is primarily a result of a combination of an estimated two–nine genetic mutations (multi-hit combinations). Although a body of research has identified hundreds of cancer-causing genetic mutations, we don't know the specific combination of mutations responsible for specific instances of cancer for most cancer types. An approximate algorithm for solving the weighted set cover problem was previously adapted to identify combinations of genes with mutations that may be responsible for individual instances of cancer. However, the algorithm's computational requirement scales exponentially with the number of genes, making it impractical for identifying more than three-hit combinations, even after the algorithm was parallelized and scaled up to a V100 GPU. Since most cancers have been estimated to require more than three hits, we scaled out the algorithm to identify combinations of four or more hits using 1000 nodes (6000 V100 GPUs with $\approx 48\times10^6$ processing cores) on the Summit supercomputer at Oak Ridge National Laboratory. Efficiently scaling out the algorithm required a series of algorithmic innovations and optimizations for balancing an exponentially divergent workload across processors and for minimizing memory latency and inter-node communication. We achieved an average strong scaling efficiency of 90.14% (80.96% – 97.96% for 200 to 1000 nodes), compared to a 100 node run, with 84.18% scaling efficiency for 1000 nodes. With experimental validation, the multi-hit combinations identified here could provide further insight into the etiology of different cancer subtypes and provide a rational basis for targeted combination therapy.

I. Introduction

Cancer is one of the leading causes of death in the US, accounting for 21% of all deaths, second only to heart disease [1]. Clearly, there is a need for more effective treatments. Cancer is a polygenic disease, resulting from a combination of multiple genetic mutations (multi-hit combinations) [2], [3].

The overall goal of this work is to identify the most likely combinations of genes with mutations that may be responsible for individual instances of cancer. This approach is unlike most existing computational approaches that search for individual cancer genes in which mutations increase the risk of cancer [4]–[10]. Although there are many contributing factors to the genesis and progression of cancer, other than genetic mutations, such as tumor micro-environment [11], epigenetic modifications [12], copy number variations [13], etc., this work is limited to identifying combinations of genes with mutations that may play a role in carcinogenesis or cancer progression.

An algorithm based on an approximate solution to the weighted set cover problem was previously developed to identify multi-hit combinations of genes with mutations that may be responsible for individual instances of cancer [14]–[16]. While there are several parallel algorithms [17], [18] for the set cover problem with a comparable approximation ratio (accuracy), we selected and extended the one [19] that best mapped to our formulation of the problem: enumerating a set as a combination of a given length with set-weight calculated from input data associated with each combination.

However, the computational complexity of the algorithm scales exponentially with the number of hits, i.e. G^h, where $G \approx 20000$ is the number of genes and h is the number of hits. This exponential scaling limits the number of hits that can be practically identified to three-hit combinations, for sample size greater than 200, even with parallelization across thousands of processors on an NVIDIA V100 graphical processing unit (GPU) [16]. For example, the identification of three-hit combinations took 13860 minutes on a single CPU and 23 minutes on a single GPU [16]. We estimate that the identification of four-hit combinations would require over 500 years on a single CPU and over 40 days on a single GPU.

Previous studies have shown that most cancers require an estimated four – nine hits, on average [3], [20]–[22].

We previously estimated that 11 of 17 cancer types studied required four or more hits [3]. Therefore, two- and three-hit combinations will not be able to identify the specific combination of gene mutations responsible for individual instances of most cancers, which is the ultimate goal of this approach.

To be able to identify combinations of more than three-hits, we restructured and optimized the algorithm for parallel execution across multiple GPUs on multiple nodes of the Summit supercomputer at the Oak Ridge National Laboratory. Efficiently scaling out the algorithm for parallel execution across thousands of GPUs presented three key challenges: balancing the load across millions of GPU processing cores, minimizing the use of slow global memory, and minimizing inter-processor and inter-node communication. We describe our approach for addressing these challenges in Section III. This approach resulted in an estimated 7192-fold speedup using 6000 V100 GPUs (1000 Summit nodes), compared to the execution on a single GPU, for identifying four-hit combinations, as described in the Results section. In Section V, we describe limitations of our approach and how they might be addressed. Our findings are summarized in Section VI.

A. Our contributions

We report four contributions through this manuscript.

1) We developed an approximate weighted set cover (WSC) algorithm to identify multi-hit combinations of carcinogenic genes from a combinatorially complex search-space and scaled out the algorithm for parallel execution on a thousand nodes of Summit supercomputer to achieve an average strong scaling efficiency of 90.14%.
2) The WSC algorithm only needs to process the upper tetrahedral portion of three dimensional matrices, leading to half the threads being idle. We mapped the upper tetrahedral portion of the matrix to a linear thread index to eliminate idle threads.
3) Although the above eliminated idle threads, the workload within each thread varied exponentially from O(1) to O(G^2), where G is the number of genes. We developed a workload scheduler that efficiently allocates threads to GPUs such that each GPU has approximately equal workload.
4) Through a combined strategy of hierarchical data structure design and multi-kernel, multi-level parallel reduction, we reduced our memory requirement from petabytes to gigabytes. This strategy also reduced irregular memory access, memory migration between host and device, and inter-node communication overhead.

II. Related Work

Existing computational approaches for identifying carcinogenic mutations are primarily based on evaluating mutational characteristics in tumors relative to an expected baseline [2], [4]–[8], [23]. Carcinogenic mutations, thus identified, have been shown to be associated with an increased risk of the disease. However, in general they do not result in cancer by themselves. Most cancers require some combination of additional carcinogenic mutations.

A. Existing approaches for identifying carcinogenic mutations

Existing approaches compare some combination of the following characteristics in tumor samples to a presumed baseline, to identify likely driver mutations. Genes with a mutation rate that is significantly higher than an estimated baseline mutation rate are considered to be probable cancer genes [2]. Copy number variations [23], expression levels [8], gene function [10], interaction networks/pathways [8], and structural information [9] are also considered in identifying potential cancer genes. The estimated background mutation rate is generally normalized for gene size, CpG content, and synonymous mutation rate [4], [5]. Specific mutations within these cancer genes are further evaluated for recurrence, functional impact, protein sequence clustering and expected pattern of mutations, to determine which mutations are likely driver mutation [2], [6], [24]. However, none of these approaches attempt to identify combinations of mutations that are likely to be responsible for individual instances of cancer, as the following multi-hit algorithm does.

B. The multi-hit algorithm for identifying carcinogenic gene combinations

The approach presented here, unlike existing approaches, is designed specifically to identify multi-hit combinations of mutations. In our prior work [15], we mapped the problem to the weighted set cover (WSC) problem and proposed an approximate algorithm to solve it. The goal of this algorithm is to identify a set of multi-hit combinations that are frequently present in tumor samples but rarely in normal samples. The idea being that these combinations are likely to be oncogenic. Based on the observation that at least one multi-hit combination should be present in every tumor samples, i.e. "cover" the set of tumor samples, we mapped this problem to the Weighted Set Cover (WSC) problem [15].

WSC is an NP-Complete problem with no known polynomial time exact solution [25], [26]. The computational complexity for this problem is $O(2^M)$, which is the number of all possible sets of multi-hit combinations, where $M = \binom{G}{h}$ is the number of all possible h-hit combinations, $G \approx 20000$ is the number of genes and h is the number of hits. For four-hit combinations, $M \approx 7E15$, making the problem computationally intractable. However, there are approximate greedy algorithms that reduce computational complexity to $O(M \times N_t)$ making the problem tractable for small h, where N_t is the number of tumor samples.

We previously developed an approximate algorithm to solve this mapped problem [15]. Briefly, the algorithm iterates through the following steps until all tumor samples have been excluded:

1) Enumerate all possible h-hit gene combinations and compute F (weight)
2) Choose the combination with maximum F

3) Exclude all tumor samples containing mutations in the gene combination, from further consideration

The weight F is defined as follows:

$$F = \left(\frac{\alpha TP + TN}{N_t + N_n}\right) \quad (1)$$

Here, TP (True Positives) is the number of tumor samples containing mutations in the gene combination, TN (True Negatives) is the number of normal samples that do not contain mutations in the gene combination, N_t is the total number of tumor samples, N_n is the total number of normal samples, and $\alpha = 0.1$ is a penalty term to offset the algorithm's inherent bias towards true positives relative to true negatives. See Ref [15] for details.

C. Restructuring and optimizing for parallel execution on a single GPU

The use of the greedy algorithm described above made it possible to identify two-hit combinations on a single processor [15]. However, due to its exponential scaling, the algorithm was impractical for more than two-hit combination. To identify three-hit combinations in a practical time-frame (< 15 days), the sequential algorithm was restructured and optimized for parallel execution across thousands of processors on an NVIDIA V100 GPU [16]. Two key optimizations included representing the input data in a compressed binary format and mapping the upper triangular matrix of gene combination to a linear index, as summarized below.

The input data for the algorithm consists of two binary gene-sample matrices, one for tumor samples and another for normal samples. Each element of the matrix is 0 or 1 depending on whether the corresponding sample has mutations in the corresponding gene. These matrices were compressed into a binary representation with 64 samples being grouped into a single unsigned long long int variable. This compressed binary representation resulted in a 32x reduction in memory utilization for these matrices. In addition, the number of arithmetic operations were reduced through the use of bitwise operations for counting the number of samples containing mutations in a given combination of genes. Another important benefit of using bitwise operations was that it reduced the number of divergent branching operations, resulting in improved processor utilization in a GPU [27]. See Ref [16] for details.

For two-hit combinations, all possible combinations of G genes are represented by the upper triangular portion of a $G \times G$ matrix. A naive parallel implementation assigned each element of the matrix to a separate thread which then computed the value of F for the combination i and j. However, since the matrix is symmetrical along the diagonal, half of the work is redundant. It results in unbalanced workload since half of the threads are idle. By mapping the upper triangular matrix to a linear index λ [28], the idle threads where $j < i$ were eliminated, reducing the unbalanced workload as shown in Algorithm 1.

This mapping also reduces the workload imbalance for combinations of more than two hits. See Ref. [16] for details.

Algorithm 1 Computing 3-*hit* combinations with sequential mapping of upper triangular matrix (two `for` loops)

```
for λ = 1 → C(G,2) do
    j ← ⌊√(1/4 + 2λ) + 1/2⌋
    i ← λ − j(j − 1)/2
    for k = j + 1 → G do
        Compute F_{i,j,k} for combination (g_i, g_j, g_k)
        ...
```

III. Identifying combinations of more than three hits

Despite the optimizations described in the previous section, 4-hit combinations for most cancer types could not be identified in a practical timeframe (< 15 days), on a single GPU. Therefore, in this work we scaled out the algorithm by restructuring and optimizing computational steps of the algorithm, and by developing a scheduler for balanced workload distribution across GPUs and nodes of the Summit supercomputer at the Oak Ridge National Laboratory (ORNL).

A. Distributing workload across multiple nodes and GPUs

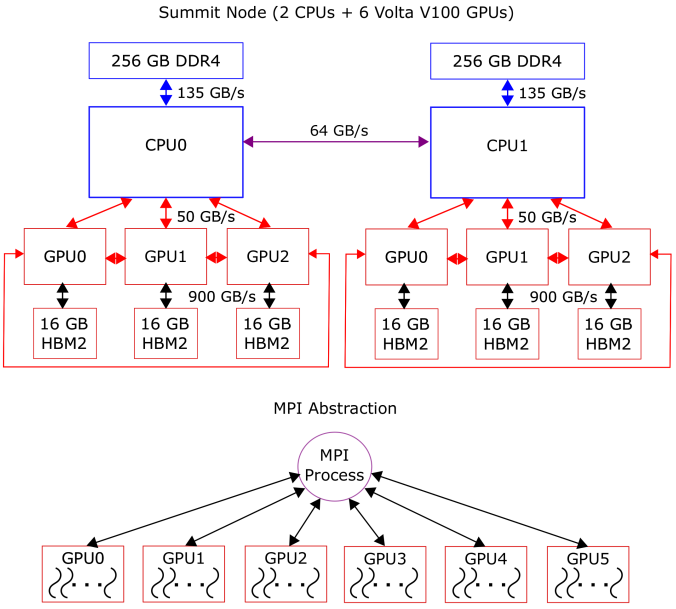

Fig. 1: Summit node as a computational unit and its abstraction with a single MPI process per node. Top: Each Summit node consists of two IBM Power9 CPUs and six NVIDIA V100 GPUs. Bottom: Each Summit node is assigned to a single MPI process along with a range of threads (curved lines) that are in turn assigned to individual processors within the GPUs.

Each Summit node has two IBM Power9 CPUs and six NVIDIA V100 GPUs. For simplicity, we abstract each node as having one CPU core that uses six V100 GPU devices, and each GPU device can serve thousands of threads (Fig. 1). For identifying four-hit combinations, a sequential implementation of the algorithm would iterate over nested `for` loops of depth

four to examine all $\binom{G}{4}$ combinations. Two–four of the four outer `for` loops can be flattened into a single `for` loop. The outermost `for` loop can then be parallelized with one thread processing one iteration of the outermost `for` loop. Depending on the number of outer loops flattened, we define four parallelization schemes. We implemented two of these schemes and we present our analysis of their scaling efficiency.

1) **1x3 scheme:** Launch G threads, each thread will run a nested loop of depth 3.
2) **2x2 scheme:** Flatten outer two loops, launch $\binom{G}{2}$ threads, each thread will run a nested loop of depth 2.
3) **3x1 scheme:** Flatten outer three loops, launch $\binom{G}{3}$ threads, each thread will run one for loop.
4) **4x1 scheme:** Flatten all four loops, launch $\binom{G}{4}$ threads, each thread will process only one combination.

We define workload as the number of combinations assuming each combination yields same number of arithmetic operations. The first scheme offers a small number of threads (limited parallelization) with heavy workload per thread. The fourth scheme offers astronomically large threads with constant operation. So, we did not consider these two. We first implemented the 2x2 scheme and built the 3x1 scheme from the lessons learnt from the former scheme.

The entire workload $\left(\binom{G}{2}\text{ or }\binom{G}{3}\text{ threads}\right)$ is distributed across hundreds of Summit nodes using the message passing interface (MPI) where each node serves one MPI process.

Algorithm 2 Computing 4-*hit* combinations with sequential mapping of upper triangular matrix (2x2 scheme, 3 `for` loops)

for $\lambda = 1 \rightarrow \binom{G}{2}$ **do**
  $j \leftarrow \lfloor \sqrt{1/4 + 2\lambda} + 1/2 \rfloor$
  $i \leftarrow \lambda - j(j-1)/2$
  for $k = j+1 \rightarrow G-1$ **do**
    for $l = k+1 \rightarrow G$ **do**
      Compute $F_{i,j,k}$ for comb (g_i, g_j, g_k, g_l)
      ...

B. Reducing workload imbalance across threads

The workload for each thread (λ) can be represented by the number of gene combinations of (g_k, g_l) processed in the inner two (one) `for` loops in Algorithm 2 (Algorithm 3). The workload per thread decreases exponentially with thread ID (Fig. 2). As the figure shows, for $G = 10$, there is considerable difference between the first and last thread's workload. This difference, which is $\binom{G}{2}$ for the 2x2 scheme, limits scaling efficiency. To reduce this difference in workload between threads, we mapped the upper tetrahedral matrix, $i \leq j \leq k$ to a linear thread ID, as shown in Algorithm 3 (3x1 scheme). This mapping spreads the workload across a larger number of threads ($\binom{G}{3}$), such that the difference between the first and last thread is reduced to G (Fig. 2).

C. Reducing workload imbalance across GPUs

Although the mapping in Algorithm 3 reduces workload difference between the first and last thread from O(G^2) to

Algorithm 3 Computing 4-*hit* combinations with sequential mapping of upper tetrahedral matrix (3x1 scheme, 2 `for` loops)

for $\lambda = 1 \rightarrow \binom{G}{3}$ **do**
  $q \leftarrow (\sqrt{729\lambda^2 - 3} + 27\lambda)^{1/3}$
  $k \leftarrow \lfloor (q/3^2)^{1/3} + 1/(3q)^{1/3} - 1 \rfloor$
  $T_z \leftarrow k(k+1)(k+2)/6$
  $\lambda' = \lambda - T_z$
  $j \leftarrow \lfloor \sqrt{1/4 + 2\lambda'} - 1/2 \rfloor$
  $i \leftarrow \lambda' - j(j+1)/2$
  for $l = k+1 \rightarrow G$ **do**
    Compute $F_{i,j,k}$ for comb (g_i, g_j, g_k, g_l)
    ...

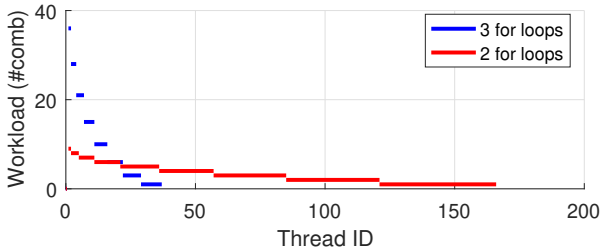

Fig. 2: Thread workload distribution for sequential mapping of upper triangular ($i \leq j$) (Algorithm 2, 2x2 scheme) and upper tetrahedral (Algorithm 3, 3x1 scheme) matrices. Tetrahedral mapping spreads out the computation across a larger number of threads, thus reducing workload imbalance between threads. Workload calculated for number of genes $G = 10$.

O(G), with $G \approx 20000$ the remaining difference still has a significant effect on scaling efficiency. The workload $\left(\binom{G-j}{2}\right)$ vs λ shows an exponential curve with reducing amount of workload with increasing global thread id λ (Figure 3(a)). A naive implementation assigns equal number of threads to each node (and each GPU), which we refer to as equi-distance (ED) scheduling. The area under this curve for each partition represents the total work per GPU. From Figure 3(a), we can see that areas under these different curves are very different. This will create significant load imbalance across MPI processes. To balance the workload we developed an alternate scheduling approach, which we refer to as equi-area (EA) scheduling. We partition the workload across MPI processes and their GPUs based on the area under each partition's curve (Figure 3(b)), resulting in a more balanced workload across GPUs (Figure 3(c)). However, efficiently determining the range of threads for each GPU is a non-trivial problem.

The objective of the equi-area scheduler was to sequentially assign threads to GPUs such that the cumulative workload is approximately equal to the average workload per GPU. A naive way to determine the partitioning of threads would be to sequentially go through each thread and accumulate workload until the average workload per GPU is reached. With $\binom{G}{3}$ threads, this approach takes tens of hours and the system

runs out of memory to store the intermediate results using a single node. We designed a more efficient method by taking advantage of the G discrete and sequential workload levels for each thread (Fig. 2). Knowing the number of threads per workload level $\left(\binom{k}{3}\right)$, where $1 \leq k \leq G-2$ is the workload level, we were able to sequentially determined the number of threads from each level to assign each GPU in O(G) operations. This approach takes less than a minute to compute the entire schedule using similar resources as before.

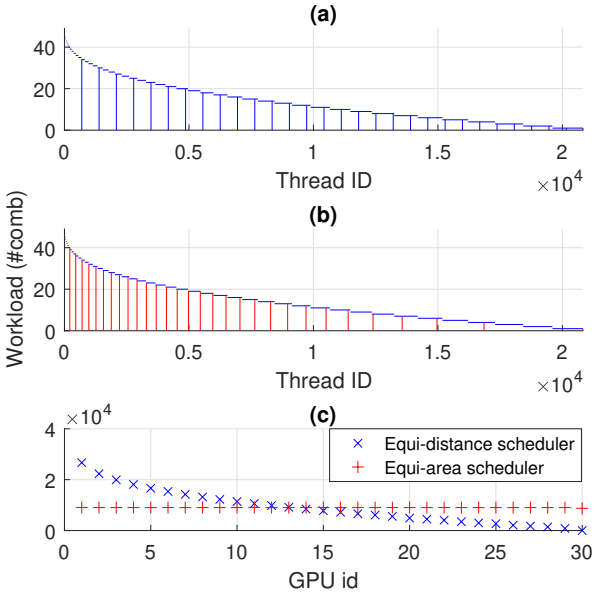

Fig. 3: Workload distribution per GPU for G = 50 and 5 nodes (30 GPUs). (a) Work (number of combinations) processed by each thread. Vertical lines show equi-distance partitioning of threads (λ) across GPUs. (b) Partitioning for equi-area scheduling where threads are assigned to nodes so that each GPU has approximately the same workload. (c) Workload for each GPU based on equi-distance and equi-area partitioning.

D. Reducing global memory access for mutation-sample data

In Algorithm 2 and 3, i, j, k, l correspond to genes in the gene-sample matrices, which reside in global memory. Reading data for these genes from global memory can stall the threads while waiting for data transfer. To reduce the number of global memory accesses, we implemented three different memory optimizations:

1) **MemOpt1:** prefetch memories corresponding to genes i
2) **MemOpt2:** prefetch memories corresponding to genes j
3) **BitSplicing:** splice out covered samples from the tumor gene-sample matrix, after every iteration of the algorithm

Each thread in Algorithm 3 corresponds to a single unique value of λ which corresponds to a single unique combination of i, j and k . Although the values of l vary within a thread, with $i < j < k < l < G$, the values for i, j and j are fixed for a thread. So, instead of repeatedly accessing matrix rows corresponding to genes i, j and k within the loop from slower global memory, we prefetch those rows into the thread's faster local memory. Pre-fetching data for i and j (MemOpt1 and MemOpt2) reduces the global memory access during computation, and the potential for processor stall while waiting for data to be retrieved from global memory.

Algorithm 3 shows one iteration of the algorithm for identifying one of multiple multi-hit combinations. Covered tumor samples (samples that contain the combination with maximum value for F) are excluded from further consideration in subsequent iterations (Sec. II-B). These tumor samples can be spliced out of the gene-sample matrix, reducing the size of the matrix and eliminating unnecessary memory accesses. Combinations identified in earlier iterations tend to exclude a large number of tumor samples, so, BitSplicing can reduce the number of columns in the gene sample matrix. With every 64 samples excluded, the number of bitwise `AND` operations are reduced by three. Reduced column width of gene sample matrix effectively reduces the number of bitwise `AND` operations linearly. In the later iterations, when fewer samples gets excluded by each iteration's best combination, BitSplicing reduces the matrix size at a slower rate.

E. Reducing requirement for unified memory and inter-node communication

We defined a structure to store a four-hit combination and its weight (F-max). The structure consists of four integer values for four gene ids and one floating point value for F-max, for a total of 20 bytes per combination. We maintain a list of these structures to find the best combination. For large G (e.g. G = 19411 for breast invasive carcinoma (BRCA)), the list consists of 1.22×10^{12} entries requiring 24.34 terabytes of memory to store 4-hit combinations.

Since each GPU has only 16GB global memory and each node had only 512GB of CPU memory, it is necessary to distribute the list across multiple nodes or GPUs. For example, using unified memory for the list can be distributed across 47 nodes. However, accessing this memory will require moving data across nodes, which combined with irregular memory access patterns will reduce computational speed significantly.

To reduce memory requirement and inter-node communication associated with the list, we implemented a multi-stage, multi-kernel parallel reduction process for calculating the global F-max value. Our implementation has two kernels: maxF kernel computes F-value for all the combinations and parallelReduceMax kernel performs a multi-stage reduction. In the previous 3-hit implementation (Algorithm 1) we performed parallel reduction (Sec. II-B) only at the second kernel, which required larger intermediate memory. To reduce the number of combinations that needed to be stored, we perform a single-stage parallel reduction in the first kernel and only stored a single combination for each block. With a block size of 512, this reduced the size of the list by a factor of 512 from 24.3

terabytes to 47.5 GB which fits into the available memory in each node. In the second kernel, we perform a multi-stage parallel reduction for all the blocks within each MPI process (GPU). Each MPI process then returns a single combination (20 bytes) to the master (Rank 0) process instead of a large list of combinations. The master process can then perform a reduction on a much smaller number of values, corresponding to the number of MPI processes.

F. Mapping global thread id λ to i, j, k

λ is stored in a 64-bit unsigned long long int variable. However, an intermediate step requires computing $A = \sqrt{729\lambda^2 - 3}$, which requires 128-bit arithmetic. Without built-in support for 128-bit operations, we compute the quantity indirectly by first computing the logarithm and then the exponential using the expression $A = exp\Big(0.5 \times \big(\log(double(3\lambda)) + \log(double(243\lambda - 1.0/\lambda))\big)\Big)$.

G. Training and test datasets

Gene mutation data in mutation annotation format (MAF), for 31 cancer types, were downloaded from the cancer genome atlas (TCGA) and summarized for input to the multi-hit algorithm. Mutations were called using the Mutect2 protocol. The cancer types, sample sizes, and summarized data associated with these cancers are included in Supplementary Information. 75% of these samples were randomly selected for the training set, with the remaining 25% being the test set. Of the 31 cancers, 11 were previously estimated to require four or more hits for oncogenesis [3]. These 11 cancer types were used for this study.

H. Experimental setup

To measure runtime performance, we performed two runs for each large node configuration (100-1000 Summit nodes at 100 nodes increment) and reported the average runtime. For measuring various performance metrics, we used NVIDIA's profiling tool NVPROF.

IV. Results

With the optimizations described above, we achieve an average strong scaling efficiency of 90.14% for 200–1000 Summit nodes, compared to 100 nodes, allowing us to identify 4-hit combinations for the 11 cancer types estimated to require four or more hits [3]. For the analysis presented below we used the cancer type with the largest dataset, breast invasive carcinoma (BRCA) with 911 tumor samples, even though this cancer type was estimated to require only two–three hits. This larger dataset gives a more accurate estimated for scaling efficiency, while allowing us to identify potential bottlenecks or limitations that may not be evident with a smaller datasets.

A. Scaling efficiency

Strong scaling efficiency (ideal runtime / actual runtime) for BRCA ranged from 80.96% – 97.96% depending on number of nodes, with a scaling efficiency of 84.18% for 1000 nodes and an average scaling efficiency of 90.14% for 200 to 1000 nodes (Fig. 4(a)). The Summit resource allocation system limits the maximum runtime to 2 hours when less than 100 nodes are used. With less than 100 nodes, the runtime exceeded 2 hours, therefore we used 100 nodes as the baseline for scaling efficiency calculations.

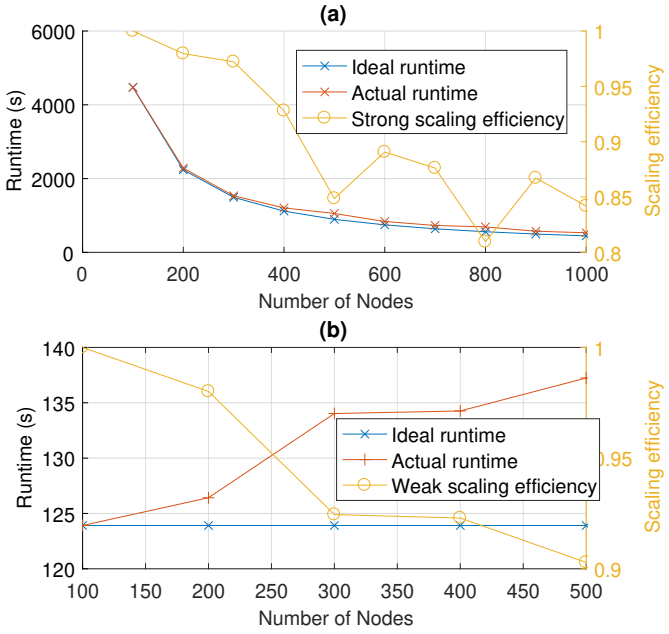

Fig. 4: Scaling efficiency for 3x1 scheme. (a) Strong scaling from 100 nodes (600 GPUs) to 1000 nodes (6000 GPUs) for the BRCA dataset. Strong scaling efficiency is 84% for 1000 nodes relative to a baseline of 100 nodes. (b) Weak scaling from 100 node (600 GPUs) to 500 nodes (3000 GPUs). Weak scaling efficiency is 90% for 500 nodes.

The strong scaling described above represents the effect of increasing the number of resources (GPUs), on runtime, for a fixed total workload. Weak scaling efficiency shows the effect of increasing the number of GPUs, on runtime, for a fixed workload per processor. The equi-area scheduler assigns an equal amount of work to each processor. However, to ensure a fixed workload per processor we limited the runs to the first iteration (Section II-B), since depending on the number of nodes used, the later iterations produce varying amount of workload. The average weak scaling efficiency for BRCA is 94.6% for 200 to 500 nodes (Figure 4(b)).

B. Contribution of optimizations to reduction in runtime

We evaluated the effect of the three memory optimization strategies, described in Section III-D, using the BRCA dataset for the three-hit algorithm running on a single GPU. Together, prefetching data for samples associated with gene i (MemOpt1), prefetching data for samples associated with gene j (MemOpt2) and splicing out data for samples associated with covered gene combinations (BitSplicing), result in a 3-fold speedup (Fig. 5).

Based on a test of the four-hit algorithm (2x2 scheme) for BRCA, equi-area scheduler (EA) achieves a 3x speedup over

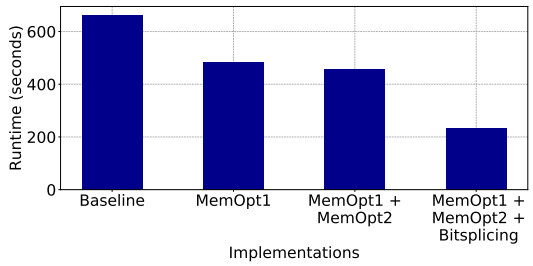

Fig. 5: Effect of three memory optimizations on runtime.

equi-distance scheduler (ED), by virtue of its more balanced workload, as described in Section III-A. The runtimes for the ED and EA schedulers were 13943 s and 4607 s respectively, for 100 node runs.

C. Compute utilization and analysis of its variance across GPUs for 2x2 scheme

The equi-area scheduler distributes approximately the same amount of workload, as measured by the number of combinations processed, across MPI processes, which are served by different nodes. The workload distributed among GPUs within each node for different MPI processes is also approximately the same. However, individual threads within a GPU and across GPUs will have different workloads. This combined with different memory access patterns for different GPUs resulted in a less than ideal strong scaling efficiency (77% for 100 nodes compared to 50 nodes).

We analyzed compute utilization for the 600 GPUs in a 100-node run for the cancer type with the smallest dataset, adenoid cystic carcinoma (ACC) (Fig. 6), to identify the reason for the low scaling efficiency. We used metrics on DRAM read/write throughput and instruction issue efficiency to analyze the variance of compute utilization cross GPUs. In general, utilization decreases with the increasing GPU index, with spikes in utilization around GPU #372, #504, and #560. GPUs with lower utilization represent processors that have completed their assigned work faster and are idle while the first GPU, with 100% utilization, is still running. Our analysis shows that compute utilization primarily depends on memory read/write throughput. Additional strategies for reducing memory access latency are discussed in Section 5.

1) DRAM read/write throughput: Figures 6(a) and (b) show that compute utilization up to GPU #500 is inversely correlated with DRAM read/write throughput. Although each GPU is assigned approximately the same workload, the range of memory accessed by threads within those GPUs decreases exponentially. For example, the thread with thread-id $\lambda = 0$ accesses $G \approx 20000$ different memory locations, while the thread with $\lambda = \binom{G}{2} - 2$ accesses only 3 memory locations. DRAM read/write thoughput is an indication of the number of cycles required for successful memory accesses. Above GPU #500 read/write throughput continues to increase without a corresponding decrease in utilization, indicating a transition of processor bottleneck from being memory bound to being compute bound. During this transition read/write throughput still affects utilization but to a smaller extent. This is reflected in the smaller spikes in utilization corresponding to spikes in read/write throughput for GPU index > 500.

2) Instruction issue efficiency: To further understand how DRAM read/write throughput affects compute utilization, we analyzed instruction issue efficiency. Thread blocks assigned to a GPU are assigned to its streaming multiprocessors (SM) and their execution is scheduled in groups of 32 threads (warp). The execution of these warps can be stalled if all necessary resources are not available or if all dependencies have not been satisfied. A breakdown of the stalled cycles shows three major contributors: memory dependency, memory throttle, and execution dependency (Fig. 6(c)). Stalls due to memory dependency indicate that resources required for load/store from memory are not available. Stalls due to memory throttle indicate that excessive pending memory operations are preventing further execution. Stalls due to execution dependency indicate that input data required for the instruction is not yet available. DRAM read/write throughput affects all three of these factors resulting in reduced compute utilization.

D. Compute utilization for 3x1 scheme

The scaling efficiency for the 2x2 scheme was as low as 36% for ESCA (Esophageal Carcinoma) for a 500 node run compared to a 100 node run. Our analysis from Section IV-C indicates this poor performance can be attributed to the variability in the compute utilization across nodes. To improve compute utilization by making memory access more regular, we adopted the 3x1 scheme which resulted in an average of 91.14% strong scaling efficiency. Figure 7 shows a balanced compute utilization across MPI processes for a 100-node run.

E. Communication overhead for 3x1 scheme

Due to the slight variance in the computation time of various MPI processes, the message passing overhead is hidden by the largest computation time of individual MPI processes (Figure 8).

F. Classification performance for the 4-hit combinations identified

We identified 151 4-hit combinations for 11 cancer types, using a 75% training dataset. Names of the 11 cancer types and all of the 4-hit combinations identified are listed in Supporting Information.

To evaluate the quality of the identified 4-hit combinations, we built a classifier per cancer type. The classifier measures the accuracy (sensitivity and specificity) with which the 4-hit combinations for each cancer type can differentiate between tumor and normal samples. For a given cancer type, let the set of combinations be $c_1, c_2, \ldots, c_p$. The classifier will classify a sample as a tumor sample if that sample has mutations in all the genes of any one of the combinations c_i $(1 \leq i \leq p)$. If there is no such combination, the sample will be classified as a normal sample. Using these per-cancer classifiers, we evaluate the classifiers' performance on the test dataset for each of the 11 cancer types. These classifiers achieve 83% sensitivity (95% Confidence Interval (CI) = $72 - 90\%$) and

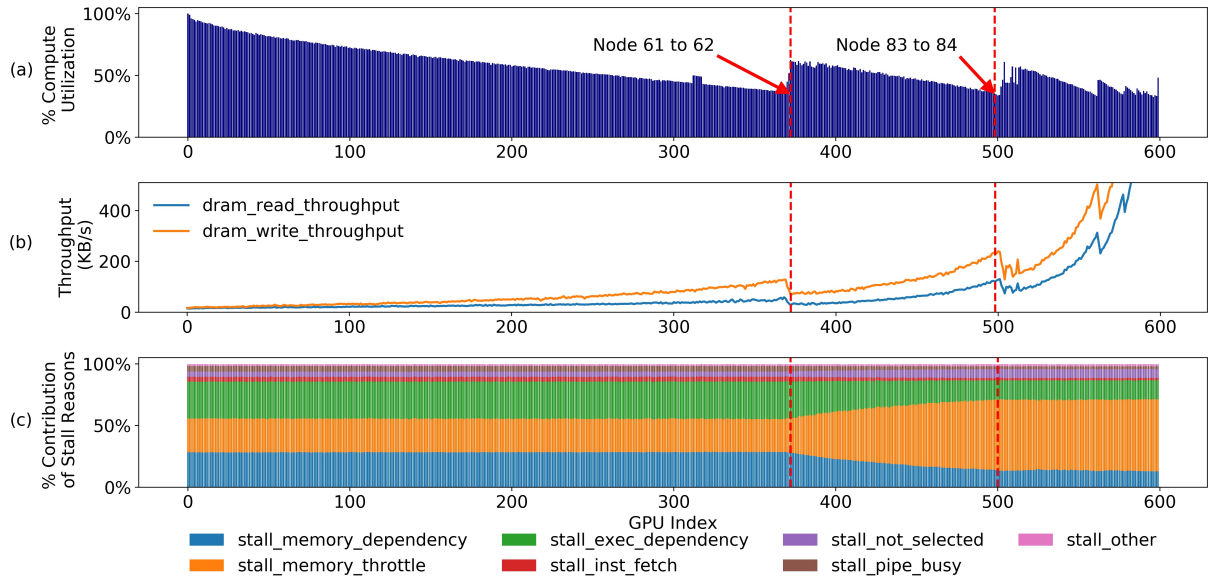

Fig. 6: Compute utilization (a) is inversely correlated with DRAM read/write throughput up to GPU #500 (b). Above GPU #500 read/write throughput increases and the processor transitions from being memory bound to being compute bound. (c) Low read/write throughput stalls warp execution while data from memory is accessed.

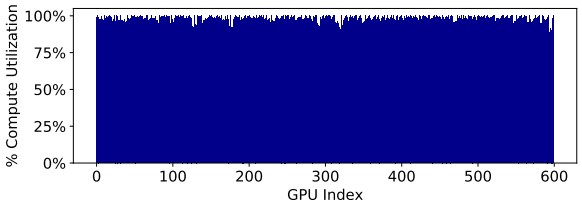

Fig. 7: Compute utilization for 3x1 scheme using BRCA dataset.

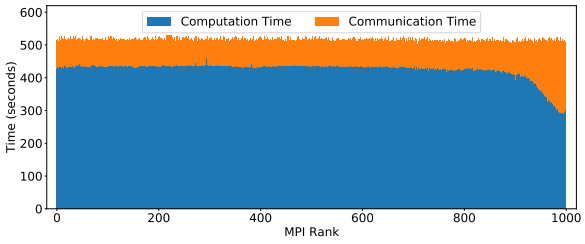

Fig. 8: Computation- and communication-time distribution across MPI processes for a 1000-node run.

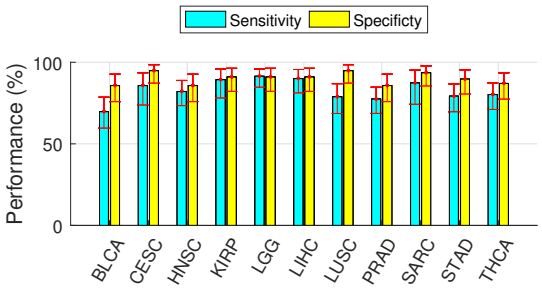

Fig. 9: Classification performance of the identified combinations. 4-hit combinations were identified using a training dataset consisting of randomly selected 75% of the available tumor and normal samples. Classification performance measured by sensitivity and specificity was based on the remaining 25% test dataset. For the 11 cancer types considered here, average sensitivity and specificity were 83% and 90% respectively. Error bars represent 95% confidence interval (CI).

90% specificity (95% Confidence Interval (CI) = $81 - 96\%$) on average (Fig. 9).

V. Discussion

The algorithm presented in this work was able to identify sets of multi-hit combinations that could differentiate between tumor and normal samples with an average 83% sensitivity and 90% specificity. However, many of the mutations in the gene combinations were correlated but not causative, i.e. passenger mutations, not driver mutations. Consider for example the top 4-hit combination for brain low grade glioma (LGG). The 4-hit combination consists of the genes IDH1, MUC6, PABPC3 and TAS2R46. The distribution of mutations for IDH1 and MUC6 in tumor and normal samples are shown in Fig. 10. Comparing the distribution of IDH1 mutations in LGG tumor samples to mutations in normal samples (Fig. 10(a)) shows that a mutation at amino acid position 132 occurs significantly more frequently in tumor samples (400 mutations in 532 tumor samples) compared to normal samples (0 mutations in 329 samples), suggesting that it may be positively selected for in tumor cells. In fact, the mutation of this particular amino acid (Arginine 132) is a known prognostic marker for gliomas [29].

Comparing the distribution of MUC6 mutations in LGG tumor samples to mutations in normal samples (Fig. 10(b)) shows no significant differences. It is likely that the mutations in MUC6 are passenger mutations that do not contribute to tumor progression.

The above example illustrates that, although our algorithm may be able to identify mutations that contribute to cancer progression (e.g. IDH1), many of the mutations in the gene combinations identified are likely to be passenger mutations (e.g. MUC6). To identify combinations of true oncogenic mutations will require searching for specific combinations of mutations within genes instead of combinations of genes with mutations.

Based on the computational complexity of the algorithm (see Section II-B), extending the 4-hit algorithm from combinations of $\sim 2 \times 10^4$ genes to combinations of $\sim 4 \times 10^5$ protein-altering mutations will require a computational speedup of $\sim 10^5$ relative to the estimated single GPU runtime for the optimized code presented above. In addition, identifying combination of each additional number of hits will require an additional speedup of $\sim 4 \times 10^5$, e.g. going from 4-hit combinations t o 5-hit combinations. An additional challenge presented by mutation level combinations is that the input mutation-sample matrices are 20 times larger than gene-sample matrices, representing increased latency due to additional global memory access requirements. Following are three of many possible strategies that can help address these challenges. (1) Parallelize execution across the 27,648 GPUs on the Summit supercomputer. (2) Distributing only the required subset of mutation-sample matrices to each GPU. (3) Limit combinations to the most probable oncogenic mutations. (4) Incorporate memory latency into the scheduling algorithm.

VI. Conclusions

Cancer is caused primarily by a combination of a small number of genetic mutations – multi-hit combinations. An algorithm based on an approximate solution to the weighted set cover problem can identify potential multi-hit combinations. However due to the exponential scaling of the computational requirement of the algorithm with the number of hits, the identification of more than 3-hit combinations proved infeasible even after parallelization across the thousands of processing cores on a NVIDIA V100 GPU. Most cancers however require an estimated four – nine hits. To identify combinations of more than three hits, we restructured and optimized the algorithm for parallel execution across multiple GPUs on multiple nodes of the Summit supercomputer at the Oak Ridge National Laboratory. The algorithmic optimizations described here resulted in a scaling efficiency of 84% for 1000 Summit nodes, compared to 100 nodes, allowing us to identify 4-hit combinations for 11 cancer types estimated to require more than 3-hits. Analysis of the multi-hit combinations identified, shows that many of the genes contain passenger mutations rather than driver mutations, indicating a need to use individual mutations within genes as the input data instead of genes with mutations. Despite this limitations, the multi-hit combinations identified do include known cancer genes, suggesting that with further refinement the algorithm has the potential for identifying combinations of cancer causing mutations.

Supporting Information

Code, data and results are available at https://bitbucket.org/sajal000/multihit-on-summit

Acknowledgements

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

This research was funded by the Edward Via College of Osteopathic Medicine (VCOM) FY20 REAP grant No. 10333

References

[1] K. D. Kochanek, S. L. Murphy, J. Xu, and E. Arias, "Deaths: final data for 2017," National Vital Statistics Reports, vol. 68, no. 9, pp. 1–76, 2019.

[2] The ICGC/TCGA Pan-Cancer Analysis Genomes Consortium and others, "Pan-cancer analysis of whole genomes," Nature, vol. 578, no. 7793, p. 82, 2020.

[3] R. Anandakrishnan, R. T. Varghese, N. A. Kinney, and H. R. Garner, "Estimating the number of genetic mutations (hits) required for carcinogenesis based on the distribution of somatic mutations," PLoS Comput Biol, vol. 15, no. 3, p. e1006881, 2019.

[4] M. S. Lawrence, P. Stojanov, P. Polak, G. V. Kryukov, K. Cibulskis, A. Sivachenko, S. L. Carter, C. Stewart, C. H. Mermel, S. A. Roberts et al., "Mutational heterogeneity in cancer and the search for new cancer-associated genes," Nature, vol. 499, no. 7457, p. 214, 2013.

[5] R. Tian, M. Basu, and E. Capriotti, "Contrastrank: a new method for ranking putative cancer driver genes and classification of tumor samples," Bioinformatics, vol. 30, no. 17, pp. 572–578, 2014.

[6] D. Tamborero, A. Gonzalez-Perez, and N. Lopez-Bigas, "Oncodriveclust: exploiting the positional clustering of somatic mutations to identify cancer genes," Bioinformatics, vol. 29, no. 18, pp. 2238–2242, 2013.

[7] F. Cheng, J. Zhao, and Z. Zhao, "Advances in computational approaches for prioritizing driver mutations and significantly mutated genes in cancer genomes," Briefings in Bioinformatics, vol. 17, no. 4, pp. 642–656, 2015.

[8] J. Xi, M. Wang, and A. Li, "Discovering mutated driver genes through a robust and sparse co-regularized matrix factorization framework with prior information from mRNA expression patterns and interaction network," BMC Bioinformatics, vol. 19, no. 214, pp. 1–14, 2018.

[9] M. H. Bailey, C. Tokheim, E. Porta-Pardo, S. Sengupta, D. Bertrand, A. Weerasinghe, A. Colaprico, M. C. Wendl, J. Kim, B. Reardon et al., "Comprehensive characterization of cancer driver genes and mutations," Cell, vol. 173, no. 2, pp. 371–385, 2018.

[10] S. K. Merid, D. Goranskaya, and A. Alexeyenko, "Distinguishing between driver and passenger mutations in individual cancer genomes by network enrichment analysis," BMC Bioinformatics, vol. 14, p. 308, 2014.

[11] A. Albini and M. B. Sporn, "The tumour microenvironment as a target for chemoprevention," Nature Reviews Cancer, vol. 7, no. 2, pp. 139–147, 2007.

[12] M. Stahl, N. Kohrman, S. Gore, T. Kim, A. Zeidan, and T. Prebet, "Epigenetics in Cancer: A hematological perspective," PLoS Genet, vol. 12, no. 10, p. e1006193, 2016.

[13] A. C. V. Krepischi, P. L. Pearson, and C. Rosenberg, "Germline copy number variations and cancer predisposition," Future oncology, vol. 8, no. 4, pp. 441–450, 2012.

[14] S. Dash, N. Kinney, R. Varghese, H. Garner, W.-c. Feng, and R. Anandakrishnan, Identifying Carcinogenic Multi-hit Combinations using Weighted Set Cover Algorithm, 2018, https://oaciss.uoregon.edu/icpp18/publications/pos138s2-file1.pdf.

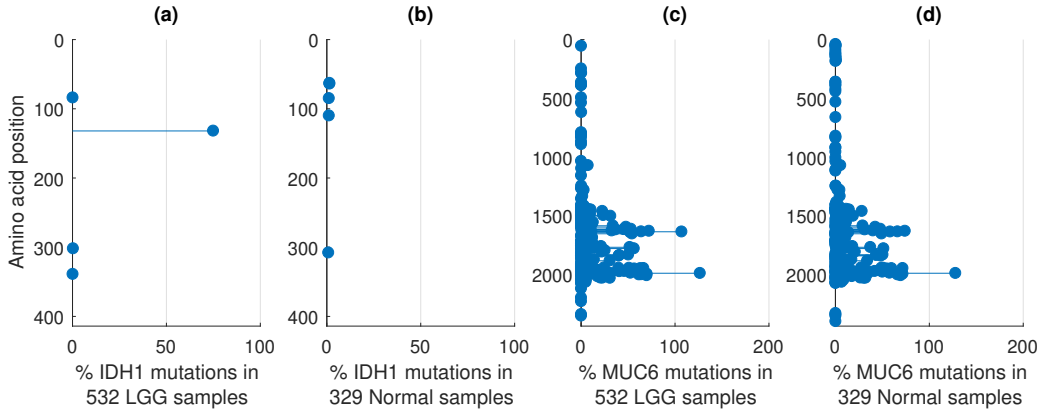

Fig. 10: Distribution of mutations in two of the genes in the top 4-hit combination in brain low grade glioma (LGG). (a) Percentage of mutations in IDH1 for LGG and (b) normal samples. (c) Percentage of mutations in MUC6 for LGG and (d) normal samples.

[15] S. Dash, N. A. Kinney, R. T. Varghese, H. R. Garner, W.-c. Feng, and R. Anandakrishnan, "Differentiating between cancer and normal tissue samples using multi-hit combinations of genetic mutations," Scientific Reports, vol. 9, no. 1, p. 1005, 2019.

[16] Q. Al Hajri, S. Dash, W.-c. Feng, H. R. Garner, and R. Anandakrishnan, "Identifying multi-hit carcinogenic gene combinations: Scaling up a weighted set cover algorithm using compressed binary matrix representation on a gpu," Scientific Reports, vol. 10, no. 1, pp. 1–18, 2020.

[17] G. E. Blelloch, H. V. Simhadri, and K. Tangwongsan, "Parallel and i/o efficient set covering algorithms," in Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures, 2012, pp. 82–90.

[18] C. Koufogiannakis and N. E. Young, "Distributed and parallel algorithms for weighted vertex cover and other covering problems," in Proceedings of the 28th ACM symposium on Principles of distributed computing, 2009, pp. 171–179.

[19] B. Al-Lazikani, U. Banerji, and P. Workman, "Combinatorial drug therapy for cancer in the post-genomic era," Nature Biotechnology, vol. 30, no. 7, p. 679, 2012.

[20] P. Armitage and R. Doll, "The age distribution of cancer and a multi-stage theory of carcinogenesis," Br J Cancer, vol. 8, no. 1, p. 1, 1954.

[21] D. Ashley, "The two" hit" and multiple" hit" theories of carcinogenesis." Br J Cancer, vol. 23, no. 2, p. 313, 1969.

[22] E. G. Luebeck and S. H. Moolgavkar, "Multistage carcinogenesis and the incidence of colorectal cancer," Proc Natl Acad Sci USA, vol. 99, no. 23, pp. 15 095–15 100, 2002.

[23] G. Ciriello, M. L. Miller, B. A. Aksoy, Y. Senbabaoglu, N. Schultz, and C. Sander, "Emerging landscape of oncogenic signatures across human cancers," Nature genetics, vol. 45, no. 10, pp. 1127–1133, 2013.

[24] M. S. Lawrence, P. Stojanov, C. H. Mermel, J. T. Robinson, L. A. Garraway, T. R. Golub, M. Meyerson, S. B. Gabriel, E. S. Lander, and G. Getz, "Discovery and saturation analysis of cancer genes across 21 tumour types," Nature, vol. 505, no. 7484, p. 495, 2014.

[25] J. Hartmanis, "Computers and intractability: a guide to the theory of NP-completeness," Siam Review, vol. 24, no. 1, p. 90, 1982.

[26] R. Anandakrishnan, "A partition function approximation using elementary symmetric functions," PloS One, vol. 7, no. 12, p. e51352, 2012.

[27] R. Anandakrishnan, T. R. Scogland, A. T. Fenley, J. C. Gordon, W.-c. Feng, and A. V. Onufriev, "Accelerating electrostatic surface potential calculation with multi-scale approximation on graphics processing units," Journal of Molecular Graphics and Modelling, vol. 28, no. 8, pp. 904–910, 2010.

[28] C. A. Navarro and N. Hitschfeld, "Gpu maps for the space of computation in triangular domain problems," in 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS). IEEE, 2014, pp. 375–382.

[29] J. Li, J. Huang, F. Huang, Q. Jin, H. Zhu, X. Wang, and M. Chen, "Decreased expression of idh1-r132h correlates with poor survival in gastrointestinal cancer," Oncotarget, vol. 7, no. 45, p. 73638, 2016.