GenomicsBench: A Benchmark Suite for Genomics*

Arun Subramaniyan, *University of Michigan*, Ann Arbor, MI, USA, arunsub@umich.edu

Yufeng Gu, *University of Michigan*, Ann Arbor, MI, USA, yufenggu@umich.edu

Timothy Dunn, *University of Michigan*, Ann Arbor, MI, USA, timdunn@umich.edu

Somnath Paul, *Intel Corporation*, Hilsboro, OR, USA, somnath.paul@intel.com

Md Vasimuddin, *Intel Corporation*, Bangalore, KA, India, vasimuddin.md@intel.com

Sanchit Misra, *Intel Corporation*, Bangalore, KA, India, sanchit.misra@intel.com

David Blaauw, *University of Michigan*, Ann Arbor, MI, USA, blaauw@umich.edu

Satish Narayanasamy, *University of Michigan*, Ann Arbor, MI, USA, nsatish@umich.edu

Reetuparna Das, *University of Michigan*, Ann Arbor, MI, USA, reetudas@umich.edu

***Abstract*—Over the last decade, advances in high-throughput sequencing and the availability of portable sequencers have enabled fast and cheap access to genetic data. For a given sample, sequencers typically output fragments of the DNA in the sample. Depending on the sequencing technology, the fragments range from a length of 150-250 at high accuracy to lengths in few tens of thousands but at much lower accuracy. Sequencing data is now being produced at a rate that far outpaces Moore's law and poses significant computational challenges on commodity hardware. To meet this demand, software tools have been extensively redesigned and new algorithms and custom hardware have been developed to deal with the diversity in sequencing data. However, a standard set of benchmarks that captures the diverse behaviors of these recent algorithms and can facilitate future architectural exploration is lacking. To that end, we present the GenomicsBench benchmark suite which contains 12 computationally intensive data-parallel kernels drawn from popular bioinformatics software tools. It covers the major steps in short and long-read genome sequence analysis pipelines such as basecalling, sequence mapping, de-novo assembly, variant calling and polishing. We observe that while these genomics kernels have abundant data level parallelism, it is often hard to exploit on commodity processors because of input-dependent irregularities. We also perform a detailed microarchitectural characterization of these kernels and identify their bottlenecks. GenomicsBench includes parallel versions of the source code with CPU and GPU implementations as applicable along with representative input datasets of two sizes - small and large.**

***Index Terms*—Genomics, Bioinformatics, Benchmarking, Computer Architecture.**

I. Introduction

Genomics is at the forefront of the precision medicine revolution. Genome sequencing can help in early cancer detection [1], developing targeted therapies to different tumor mutations [2], identifying the causes of complex genetic diseases [3], assessing risk factors, and developing new drugs. For example, 42% of the drugs approved by FDA in 2018 were based on precision medicine data obtained from genome sequencing [4]. With the advent of portable and cheap sequencers, it is now feasible to test and monitor the emergence of novel infectious diseases such as COVID-19 [5] among our population and take timely action to prevent their spread.

A genome is a long string of DNA bases or nucleotides (A, C, G and T). The human genome, for example, contains ∼6 `billion bases`, ∼3 `billion bases` per DNA strand. Genome sequencing refers to the process of determining the sequence of bases (i.e., A, C, G and T) in an individual's DNA. Genome sequencers typically read DNA by fragmenting it into billions of short substrings (called "reads").

Genome sequencing technology is far outpacing Moore's law in computing. Over the last decade, they have become increasingly cheaper, faster, more portable, and produce longer reads. The cost to sequence a human genome has dropped from \$10 million, a decade ago, to less than \$1000 today. Sequencing providers like Illumina can sequence a human genome for \$600 [6] and BGI/MGI [7] has further reduced the cost to \$100. Apart from the dramatic reductions in cost, there has also been a corresponding increase in sequencing machine throughput. For example, MGI's DNBSEQ-TX and Illumina's Novaseq 6000 produce 20 Terabases [8] and 3.3 Terabases per day respectively [9]. In addition, sequencing no longer requires large bench-top instruments. Oxford Nanopore has introduced the portable MinION sequencer which can produce longer reads (few Kilobases to Megabases) in real-time, although with a higher error rate (5-15%). These portable sequencers also enable a kind of software-defined sequencing paradigm by exposing interfaces to control the length of DNA in real-time as it passes through the pore [10]. Taken together, all these developments have given rise to widespread usage of genome sequencing and ushered in the era of population genomics with several countries/organizations aiming to sequence the genomes of millions of humans [11]–[13]. However, computing solutions, hampered by challenges in scaling transistors, have not been keeping pace.

In this paper, **we identify commonly used modern sequencing pipelines, characterize their performance, and**

This work was supported in part by Precision Health at the University of Michigan, by the Kahn foundation, by the NSF under the CAREER-1652294 award and the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.

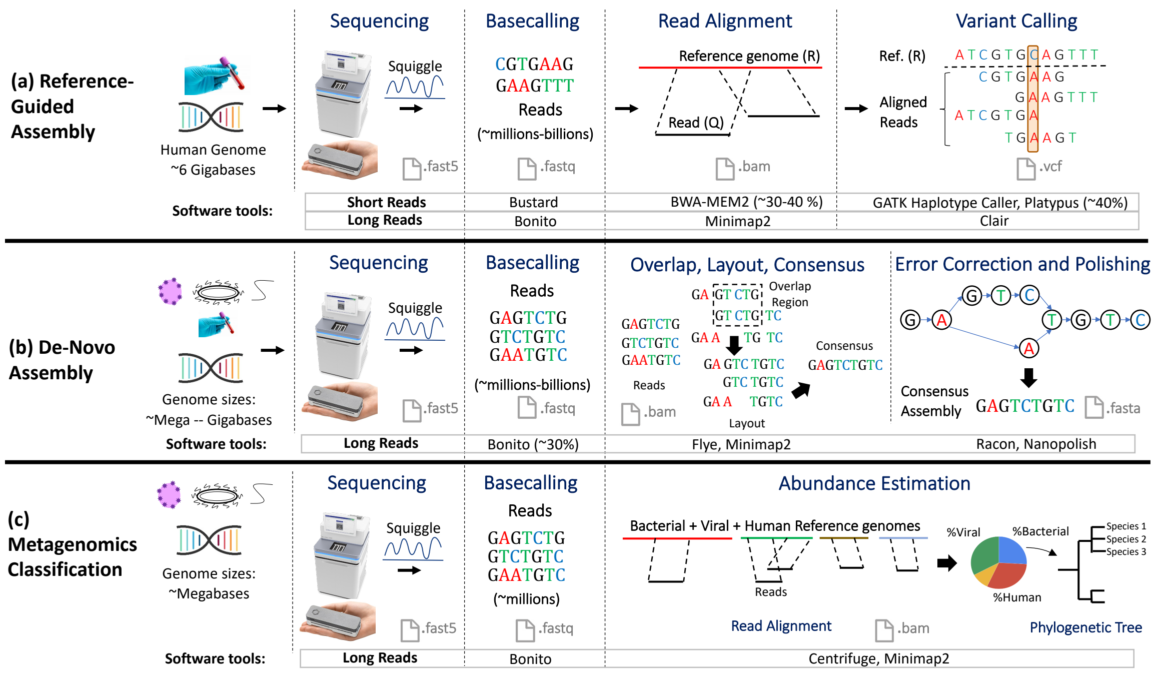

Fig. 1. Common workflows in genomics

extract their compute-intensive kernels. The goal is to compile a standardized genome sequencing benchmark suite that highlights the growing compute need in genomics and helps shape future computing research in this space. Such an effort has been lacking for this important computational domain. Some notable prior works that perform detailed architecture characterization of important bioinformatics workloads such as BioPerf [14], BioBench [15] and MineBench [16] were carried out in the last decade when sequencing technologies were still nascent and not so diverse. Modern sequencing pipelines have vastly different bandwidths, latencies, portability requirements, algorithms, and pipelines than those used a decade ago. For instance, new kernels that leverage vectorized implementations for dynamic programming are now common. Machine learning algorithms are now widely used to process long but noisy reads. There is a wide variety of sequencers that vary in terms of throughput, read length, and accuracy, to meet different medical research and clinical needs. These have resulted in a plethora of bioinformatics tools and pipelines. Without a standardized benchmark suite that represents common computational kernels, it becomes increasingly difficult to design efficient computing system and processor architectures for this rapidly emerging domain. There is also growing interest in developing custom hardware solutions for sequencing [17]. These efforts can also greatly benefit from the availability of a genomics benchmark suite.

To this end, we present the GenomicsBench benchmark suite which covers the three key classes of sequencing-based analysis: reference-guided assembly, de-novo assembly and metagenomics. To select benchmarks, we identify the most popular and well-maintained software tools used for different steps in these applications. We then extracted the time-consuming kernels in these tools and analyzed their performance characteristics. We observe that many of these kernels have a high degree of data-parallelism. But they are irregular, making it challenging for GPUs to exploit them. This motivates the need for newer architectures to exploit irregular data-parallelism, or newer vectorization friendly algorithms for these computational tasks.

To summarize, this paper makes the following contributions:

- We present the GenomicsBench benchmark suite consisting of 12 representative kernels spanning the major steps in short and long-read sequence analysis pipelines such as basecalling, sequence alignment, de-novo assembly, variant calling, and polishing.
- We perform a detailed analysis of the available parallelism in these benchmarks and observe that while these benchmarks have abundant data-parallelism, it cannot be easily exploited on commodity hardware due to significant irregularity.
- We perform a detailed characterization of the microarchitectural performance bottlenecks, memory access characteristics and thread scaling behavior of these benchmarks.
- We will open-source both the benchmark suite and input datasets for the benefit of the broader research community.

II. Background and Methodology

A. Common Genomics Pipelines

In this section we describe some common genomics pipelines to analyze short and long read sequencing data (also illustrated in Figure 1). All three pipelines start with the raw sequencer output. Given a biological sample, typically, multiple copies of the contained genome sequence are extracted and then decomposed into smaller nucleotide fragments. A sequencer reads the sequence of nucleotides in the fragments and generates raw signals based on what it reads. The first step in all the three pipelines prior to downstream analyses is the interpretation of these signals to derive reads, which are sequences of bases over the nucleotide alphabet $\{$A,C,G,T$\}$. This step is called **basecalling**. For Illumina sequencing machines, the signal data are fluorescence images which are converted into bases using a proprietary basecaller, Bustard [18]. For Oxford Nanopore (ONT) sequencers, raw signals are the current perturbations in the nanopore (e.g., in `fast5` format). Guppy [19] is ONT's proprietary basecaller software. We characterize the open-source research basecaller from ONT, Bonito [20] as part of the `nn-base` kernel (Section III), which demonstrates higher basecalling accuracy than Guppy [21].

Reference Guided Assembly: This pipeline reconstructs the sample genome by aligning reads from it to a reference genome and identifies differences in the sample (also called *variants*) compared to the reference genome. Typically, small differences, i.e., substitutions, short insertions and deletions (< 50 bases) are identified. Sufficient number of copies of the sample genome need to be sequenced to ensure random sequencing errors can be distinguished from true variations (each genome position is covered $30 - 50\times$ on average). This is especially needed for long reads from PacBio and ONT which have $5 - 15\%$ error rate per base [19], [22], resulting in input datasets of several hundreds of gigabytes. Subsequent analysis of this data can take several days on a modern multicore processor [23]. Figure 1 a. shows the two main time-consuming steps: **(1) Read Alignment**, which determines the best location for each read in the reference genome. **(2) Variant Calling**, which uses machine learning or statistics-based models to gather support for variants from aligned reads. BWA-MEM [24], [25] and GATK Haplotype Caller [26] are the most popular short-read software tools for these two steps recommended as part of GATK Best Practices [27]. These account for ∼30-40% and ∼40% time of the reference guided assembly pipeline respectively [25], [28]. We select the `fmi`, `bsw`, `phmm`, `nn-variant` and `dbg` kernels from this pipeline (Section III).

De Novo Assembly: This pipeline attempts to assemble the reads into a genome *de novo* based on read overlaps in the absence of a suitable reference. The availability of long reads for de novo assembly has greatly improved the quality of draft reference genomes. This is mainly because they can span large structural variations (e.g, > 50 bases insertions/deletions, large rearrangements) [29] and can help resolve mutations from maternal and paternal chromosomes [30]. Long read de novo assembly is typically done using the *overlap-layout-consensus* method as shown in Figure 1 b. In the overlap identification step, common seeds shared between read pairs are used to identify potential overlapping regions. In the layout step, these overlapping regions are extended into larger contiguous regions. Finally the consensus step corrects small errors in assembly. Large assembly errors are corrected in a later graph-based polishing step. For long-read assembly and polishing, Flye [31] and Racon [32] are popular software tools. Assembly of the human genome using Flye [31] and Racon [32] takes ∼4.5 days on a 64-thread server, each contributing ∼30% to the overall time [23]. Basecalling is performed using Guppy [19], ONT's proprietary basecaller, and also accounts for ∼30% of overall time [23]. We select the `chain`, `spoa`, `kmer-cnt` kernels from this pipeline (Section III).

Metagenomics Classification: The advent of portable sequencers like ONT MinION [33] has enabled several applications like real-time pathogen detection [34] and microbial abundance estimation [35] in the field. Abundance estimation involves aligning input microbial reads to a reference pan-genome (consisting of reference genomes of all bacteria, virus, fungi and humans) and later estimating the proportion of different microbes in the sample as shown in Figure 1 c. It is typically performed using software tools like Minimap2 [36] and Centrifuge [37].

III. GenomicsBench Benchmark Suite

FM-Index Search (fmi): The FM-index (Full-Text Index in Minute Space) is one of most common data-structures in aligners such as Bowtie2 [38], BWA-MEM [24], [25], SOAP3-dp [39] and metagenomics classification tools such as Centrifuge [37]. It is used to identify the locations of short matching substrings of the read (called *seeds*) in the reference genome. The FM-index is attractive because of its low memory footprint, ability to match substrings of any length and support for inexact matching (i.e., identifying seeds with a small number of edits with respect to the reference).

Figure 2 a. shows the FM-index constructed for a sample reference (R) and an example search query from the read. The FM-index consists of: (1) the *suffix array (SA)*, which contains the locations of lexicographically sorted suffixes of the reference genome `R`, (2) the *Burrows Wheeler Transform (BWT)*, computed as the last column of the sorted suffix array of the reference, (3) the *count table (C)* which stores the number of characters in `R` lexicographically smaller than a given character `c` and (4) the *occurrence table (Occ)* which stores the number of occurrences of a character up to a certain index in the *BWT* array.

The FM-index allows the backward search of a query of length ($|Q|$) in $\mathcal{O}|Q|$) iterations, with at most 2 memory lookups per iteration (one each for computing the start and end (s, e) intervals of the match). It is characterized by irregular memory accesses to the large *Occ* table (blue arrows in Figure 2 a.) and is both memory-latency and memory-bandwidth bound. Since the memory access characteristics of FM-index search are similar across different tools, we choose the optimized super-maximal exact match (SMEM) search computation in BWA-MEM2 [25] in our benchmark suite. SMEM computation uses the FM-index to find the longest exact match spanning a given position in the read.

Input Datasets: We provide small and large datasets, which are a set of 1M and 10M human reads respectively, each 151 bases long, from sample SRR7733443 [25].

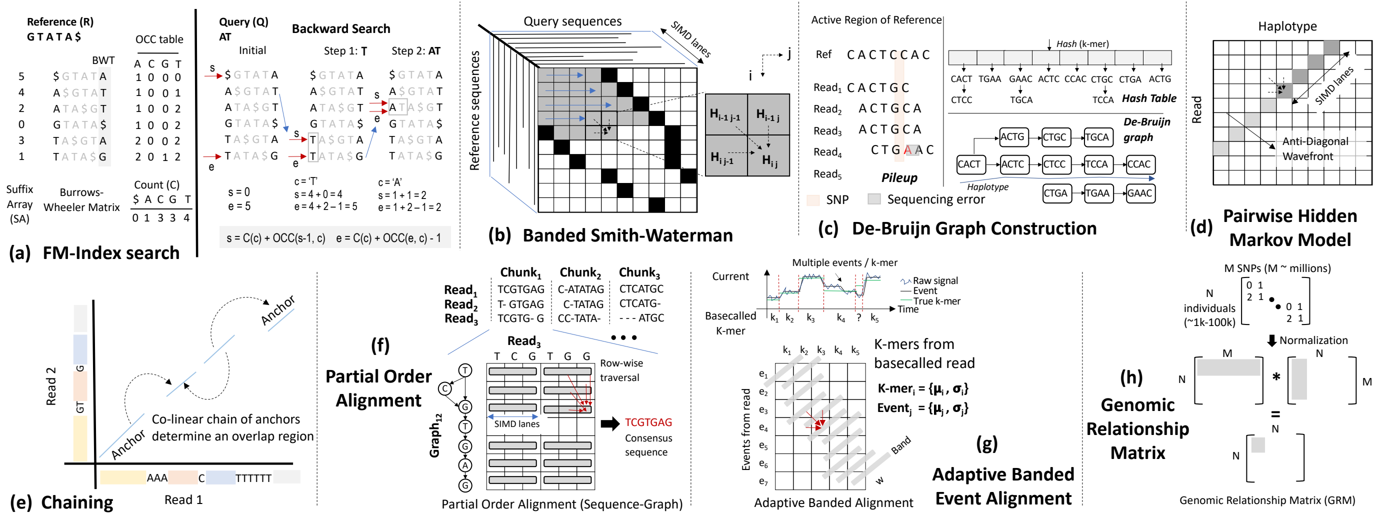

Fig. 2. Selected benchmark kernels

Banded Smith-Waterman (bsw): The Smith-Waterman algorithm [40] is a dynamic-programming algorithm that estimates the pairwise similarity between pairs of sequences X and Y with lengths m and n respectively in $\mathcal{O}(mn)$ time and space. It is commonly used in sequence alignment tools like BWA-MEM [24] and variant calling tools like GATK Haplotype Caller [26], [27] to align millions to billions of sequence pairs and is a major computation bottleneck. The similarity score for DNA sequences is typically computed using affine-gap penalties [41], which uses different penalties for different edits (i.e., substitution, insertion and deletion) and allows for identification of biologically meaningful short insertions/deletions in pairwise alignments. It requires computation of three matrices H, E and F corresponding to the different edit types. For aligning sequences with a maximum of w insertions/deletions, a banded version of Smith-Waterman is commonly used (Figure 2 b. region between the black squares) reducing time and space complexity to $\mathcal{O}(wn)$ where w is the width of the band of cells computed in each row.

$$\begin{aligned} H_{ij} &= \max\{H_{i-1,j-1} + s(i,j), E_{ij}, F_{ij}\} \\ E_{i+1,j} &= \max\{H_{ij} - q, E_{ij}\} - e \\ F_{i,j+1} &= \max\{H_{ij} - q, F_{ij}\} - e \end{aligned} \quad . \qquad (1)$$

Equation 1 shows the recurrence relation for the Smith-Waterman algorithm. $s(i,j)$ is a pre-computed similarity score between characters $X[i]$ and $Y[j]$ and the score in cell (i,j) of matrix H (i.e., H_{ij}) is the similarity score for substrings $X[0,i]$ and $Y[0,j]$. We choose the optimized banded Smith-Waterman implementation in BWA-MEM2 [25] for our benchmark suite. It makes use of inter-task parallelism to allocate similarly sized sequence pair tasks to different SIMD lanes.

Input Datasets: Our small and large datasets use 100K and 10M seed extension pairs obtained from inputs to the Smith-Waterman function in BWA-MEM2 for reads from human sample SRR7733443 [25].

K-mer Counting (kmer-cnt): A k-mer is a fixed k-length substring of a DNA sequence. K-mer counting counts the number of occurrences of each unique k-mer in the input reads. It is one of the most common tasks in bioinformatics sequence analysis and is widely used in de novo assembly [31], [42], error correction [43] and metagenomics classification [44]. Common use cases include filtering out low-frequency k-mers in the input data that are likely to be sequencing errors, finding high-frequency k-mers characteristic of repetitive genomic regions and constructing k-mer histograms to serve as signatures of the input data [45]. Typical k-mer lengths are 15-55. The computation task in k-mer counting is an incremental update to a hash-table for each k-mer. These updates can be parallelized across millions to billions of k-mers in the input dataset. We focus on shared-memory k-mer counting and characterize the k-mer counting implementation in the popular Flye assembler [31].

Input Datasets: Our small and large datasets use 1K and 50K Oxford Nanopore reads from *E.coli* sequenced by Loman lab [46].

De-Bruijn Graph Construction (dbg): Prior to calling variants using the reads aligned to a region of the reference genome (e.g., ∼100-1000 bases), it is necessary to correct read alignment artifacts. Modern variant callers like GATK Haplotype Caller [26], [27] and Platypus [47] do this by re-assembling those reads into a De-Bruijn graph and later traversing this graph to generate strings that are likely to contain variants (called *haplotypes*). The graph is constructed from both the k-mers of the read and the reference as shown in Figure 2 c. Each node in the graph represents a unique k-mer and each edge links adjacent k-mers in the input. A hash table is used to track nodes that have already been inserted into the graph. If cycles are found in the graph, graph construction is repeated by increasing the k-mer size. Each input task to this kernel is a set of reads aligned to a reference region. The re-assembly tasks can be parallelized across different regions. We model the De-Bruijn graph construction implementation in the Platypus variant caller [47] for the benchmark suite that accounts for >60% of its runtime.

Input Datasets: We use BWA-MEM aligned records from the Platinum Genomes dataset [48]. Our small dataset uses a region of chromosome 22 (bases 16M-16.5M) while the large dataset uses the entire chromosome 22.

Pairwise Hidden Markov Model (pairHMM): Using the reads aligned to a region of the reference genome and the candidate haplotypes identified from De-Bruijn graph traversal, a pairwise alignment of each read to each candidate haplotype is performed to identify the most likely haplotypes supported by the reads. The total workload per region is $|R| \times |H|$ pairwise alignments, where $|R|$ and $|H|$ are the number of reads and haplotypes respectively. Pairwise alignment is performed using a Hidden Markov Model (HMM) and the likelihood score is computed using the following dynamic-programming recurrence relations [49]:

$$\begin{array}{l} M_{ij} = (M_{i-1j-1}\theta \ + I_{i-1j-1}\kappa \ + D_{i-1j-1}\lambda) \cdot P_{ij} \\ I_{ij} = M_{i-1j}\tau \ + I_{i-1j}\epsilon \\ D_{ij} = M_{ij-1}\zeta \ + D_{ij-1}\eta \end{array} . \quad (2)$$

where: M_{ij}, I_{ij} and D_{ij} represent match, insertion and deletion probabilities for aligning read substring $R[0, i]$ to haplotype substring $H[0, j]$, where $0 \leq i \leq |R|$ and $0 \leq j \leq |H|$. These are weighted by different transition and emission parameters of the HMM: $\theta, \kappa, \lambda, \tau, \epsilon, \zeta, \eta$. P_{ij} is the prior probability of emitting bases $(R[i], H[j])$, computed using the floating point base-quality scores for the read R. Base-quality scores are typically provided by the basecaller and indicate the confidence of the basecaller in calling each base in the read. Low quality bases from the read contribute a smaller amount to likelihood score computed above. The computation in pairHMM differs from the Smith-Waterman kernel described earlier mainly in the use of floating-point computation. There exists abundant intra- and inter-task parallelism in this workload. Intra-task parallelism arises from data-parallel processing of cells along the wavefront as shown in Figure 2 d. Inter-task parallelism arises by parallel processing of different genome regions. We use the optimized SIMD implementation in GATK Haplotype Caller [26] as part of the benchmark suite and extend it to leverage inter-task parallelism using multiple CPU threads.

Input Datasets: We use the read-haplotype pair inputs to the *calcLikelihoodScore* function in GATK Haplotype Caller [26]. Our small dataset uses as input BWA-MEM aligned reads for region chromosome 22:16M-16.5M, while the large dataset uses reads aligned to the entire chromosome 22.

Chaining (chain): One of the most time-consuming steps in de novo assembly of long reads is overlap estimation between reads [31], [42]. We characterize the chaining implementation from Minimap2 [36] which is one of the most popular tools for estimating pairwise overlap between reads and extend it to support inter-task parallelism across different pairs of reads. Given a set of seeds (also called *anchors*) shared between a pair of reads, chaining aims to group together a set of co-linear seeds into a single overlapping region as shown in Figure 2 e. The chaining algorithm is a 1D dynamic programming based algorithm that compares each anchor with N previous anchors (default = 25) to determine its best parent. The recurrence relation used to estimate the maximal chaining score of the i^{th} anchor [36], [50] is:

$$score(i) = \max\Big\{ \max_{i>j\geq 1}\{score(j) + \alpha(j, i) - \beta(j, i)\}, w_i\Big\} \quad (3)$$

where w_i is the length of anchor i, $\alpha(j, i)$ is the number of matching bases between anchors i and j after accounting for overlaps between them and $\beta(j, i)$ is a penalty that is set based on the relative distance between a pair of anchors on the two reads.

Input Datasets: Our input dataset uses the anchors for 1K and 10K reads from the Pacbio sequence data for the *C.elegans* worm [50], [51] when computing overlaps with itself.

Partial-Order Alignment (poa): After assembling the reference genome of a new species, it is common to perform a *polishing* step to correct small errors in assembly using the aligned reads. Racon [52] is one of the most popular tools for long-read polishing. Given a set of reads aligned to the target genome, Racon first splits the reads into non-overlapping windows called chunks (which can be processed in parallel) and then incrementally constructs a partial-order graph [53] by aligning new sequences to it using a SIMD accelerated dynamic programming algorithm (see Figure 2 f). Later, the consensus sequence is generated from the graph using the heaviest bundle algorithm [54]. Each node in the partial-order graph represents a base of the input sequence and weighted edges represent support from different reads in the chunk. Since the nodes in multiple branches of the graph cannot be ordered relative to each other, the graph is said to be *partially ordered*. Aligning new sequences to the graph is the most time-consuming operation in Racon and has complexity $\mathcal{O}((2n_p + 1)n|V|)$, where n_p is the average number of incoming edges to nodes in the graph, $|V|$ is the number of nodes in the graph and n is the length of the read chunk. Contrast this with Smith-Waterman which has complexity $\mathcal{O}(mn)$, with regular data-dependencies. As used in Racon, our `poa` benchmark builds the consensus sequence for each chunk in a separate CPU thread.

Input Datasets: We use 1000 and 6000 consensus tasks for our small and large datasets respectively. These are obtained when polishing the Flye-assembled *Staphylococcus aureus* genome with Minimap2-aligned ONT long reads [19].

Adaptive Banded Signal to Event Alignment (abea): Comparing a time-series of raw nanopore signal data to a reference genome sequence is a common task in the polishing of long-read sequencing data and detection of methylated bases (i.e., non-standard nucleotides apart from A, C, G, T, which play an important role in controlling gene expression). After segmenting the signal data into different *events* based on sudden changes in signal current, each event is then compared against the k-mers of the reference genome using a computationally intensive dynamic programming algorithm called *adaptive banded event alignment (ABEA)* [55]. ABEA is the most time-consuming kernel when performing methylation calling using the software tool Nanopolish [56]. Event alignment is more complex than banded sequence alignment since it requires an adaptive band [57] to capture long gaps in optimal alignments especially when dealing with long and error-prone Nanopore reads. These long gaps arise because k-mers are often over-represented (up to $2\times$) by multiple events as they are sampled by the nanopore. Furthermore, event alignment uses 32-bit floating point log-likelihood computation in its scoring function and is computationally more expensive than sequence alignment. We analyze the optimized GPU implementation of ABEA [55] as part of the benchmark suite. In this heavily optimized implementation, ABEA accounts for 24.5% of total runtime.

Input Datasets: For ABEA, our small and large datasets use 1,000 and 10,000 raw FAST5 reads from chromosome 22 of NA12878, and the GRCh38 reference genome. This data was obtained from the publicly available “Nanopore WGS Consortium” dataset [58], [59].

Genomic Relationship Matrix (grm): All large-scale population genomics studies need to account for potential ancestral relationship between individuals in the study. This is done by computing a $N \times N$ matrix called Genomic Relationship Matrix (or GRM), where N in the number of individuals in the study. Each element of the GRM G_{ij} describes the average genetic similarity between individuals and is computed as follows:

$$G_{ij} = \frac{1}{S} \cdot \sum_{s=1}^{S} \frac{(x_{is} - 2p_s)(x_{js} - 2p_s)}{2p_s(1 - p_s)} \qquad (4)$$

where x_{is} and x_{js} indicate the number of copies of the non-reference base at location s for individuals i and j respectively and p_s is expected frequency of a non-reference base at location s in the population. S is the total number of SNV (Single Nucleotide Variation) location markers in the reference genome. We extract the GRM kernel from the popular population genomics software PLINK2 [60]. The kernel performs dense matrix multiplication and can benefit from parallel computation of different output elements as shown in Figure 2 h.

Input Datasets: We compute the GRM on SNV data belonging to 2504 individuals from 1000 Genomes Project Phase 3 [60]. Our small dataset uses 194K variants from chromosome 22 and our large dataset uses 1.07M variants from chromosome 1.

Neural Network-based Base Calling (nn-base): When performing nanopore based genome sequencing, raw nanopore signal data must be correctly converted to a sequence of nucleotide bases through a process called *basecalling* discussed earlier. As DNA moves through a nanopore, it does so at a highly variable rate, and the resulting current is affected by multiple consecutive nucleotides occupying the pore ($\sim$5-10, depending on the pore chemistry). Due to the limited resolution of the ADC sampler and unavoidable background noise, there is considerable overlap between current levels measured for different 5-mers. Basecallers resolve this ambiguity in two stages. First, a deep recurrent or convolutional neural network aggregates contextual information to determine the most likely nucleotide observed at each time step. Using these probabilities, a connectionist temporal classification decoder [61] then determines the most likely sequence. The neural network is by far the most time-consuming basecalling stage. In order to make this computation regular and parallelizable, existing basecallers segment the signal and perform inference on many independent chunks, stitching the final sequence together as a post-processing step. Our benchmark includes the GPU-based CNN basecaller Bonito, which currently boasts the highest basecalling accuracy [20].

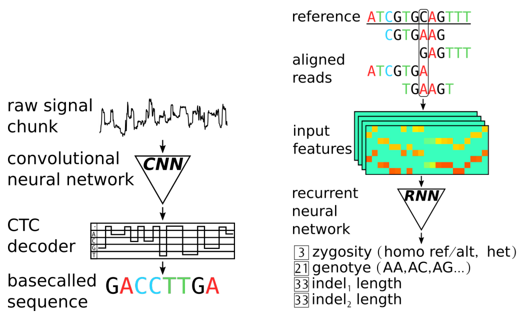

Fig. 3. Overview of Bonito (left) and Clair (right)

Input Datasets: For basecalling, our small and large input datasets are 100 and 1,000 raw FAST5 reads from chromosome 20 of NA12878. This data was obtained from the publicly available “Nanopore WGS Consortium” dataset [58].

Pileup Counting (pileup): A common pre-processing step in long-read neural network variant callers such as Medaka [62] involves parsing of alignment data for all reads aligned to a region of the reference genome (called *read pileup*) and generating counts for different bases, insertions and deletions at these different pileup locations. These counts are later analyzed by the recurrent neural network to call variants. This pre-processing step is time consuming because it involves random access into the alignment record to extract and parse alignment information (represented as a CIGAR string [63]). Fortunately, the pre-processing step can leverage inter-task parallelism by distributing the processing of different 100 kilobase regions of the reference genome to different CPU threads. The benchmark suite includes the inter-task parallel version of pileup counting.

Input Datasets: We use the results from Minimap2 alignment of ONT reads. Our small dataset uses aligned reads to the *Staphylococcus aureus* genome [19], while the large dataset uses reads aligned to chromosome 20 of sample HG002 [64].

Neural Network-based Variant Calling (nn-variant): Long-read variants callers examine the read pileup for a particular genome reference position and call homozygous and heterozygous variants with respect to that reference. We chose to analyze the Clair variant caller because it outperforms competing tools in terms of both performance and accuracy for long reads [65]. As input, Clair accepts a size $33 \times 8 \times 4$ tensor. Given a particular reference position, this tensor is generated using pileup information for 16 bases flanking each side $(16 + 1 + 16 = 33)$, and considering the pileup counts for each base (A,C,G,T) and strand (forward,reverse) individually ($2 * 4 = 8$). Furthermore, 4 different encodings of the same information is used: **(a)** raw pileup counts, **(b)** support for insertions relative to (a), **(c)** support for deletions relative to (a), and **(d)** support for alternative variants or alleles relative to (a). Clair uses a series of recurrent neural networks with bidirectional long short-term memory (LSTM) units and fully-connected layers to predict a potential variant's genotype, zygosity, and indel length of each haplotype. Refer to [65] for network details.

Input Datasets: For benchmarking Clair on long reads, we selected all raw FAST5 reads from the q13.12 region of chromosome 20 of NA12878 from the "Nanopore WGS Consortium" [58] dataset. These reads were basecalled using high-accuracy Guppy 3.6.0, and mapped using Minimap2. Our small dataset variant called the first 10,000 reference positions from this region, and our larger dataset used 500,000.

CPU	Intel Xeon E3-1240 v5 3.5 GHz; AVX2; 1 socket; 8 threads
L1 I&D cache	4 x 32KB Inst; 4 x 32KB Data, 8-way
L2 cache	4 x 256KB, 4-way
L3 cache	4 x 2 MB, 16-way
Memory bandwidth	31.79 GB/s

TABLE I
BASELINE SYSTEM CONFIGURATIONS.

IV. Performance Characterization of Benchmarks

A. Characterization Methodology

Several of the genomic analysis tools described earlier operate on large datasets and can run for several days. To keep the study manageable, we adopt the following methodology. We first profile all software tools with Intel VTune Profiler 2020 [66] as well as manual timing instrumentation to identify the most time-consuming kernels in both single and multi-thread settings. Later, we isolate these kernels and run representative input datasets of two sizes. Kernel executions with the `small` inputs finish in a few minutes, while the `large` inputs take 5–20 minutes on a single-thread. Both the `small` and `large` inputs capture the bottlenecks in the original application and exercise the kernel in similar ways (e.g., similar proportions of different dynamic instructions and memory accesses with different strides). We use the MICA pintool [67] to compute statistics on instruction distribution. Cache miss and memory stalls are obtained using performance counter events from the hardware event-based sampling collector [68]. All kernels and inputs/outputs are extracted as-is from the original software tools. The tools already support multithreading. For ease of benchmarking, we made the following modifications to the extracted benchmarks: (1) OpenMP parallelization with dynamic scheduling was used to reliably evaluate thread scaling of the benchmark after isolation from the software tool and (2) file I/O-related driver code was added for reading inputs and writing results. GPU benchmarks are characterized using Nvidia's Visual Profiler [69] and *nvprof* on the Nvidia Titan Xp GPU with 12GB GDDR5x memory. Table I details our experimental machine configuration. We present characterization results for all benchmarks except `nn-variant` which failed to complete successfully using `nvprof` on both a native run as well as within a Docker container.

[1]The proportion of runtime taken by the `pileup` kernel increases dramatically when a GPU is used for variant calling.

B. Parallelism

1) CPU benchmarks

In this section, we present a detailed characterization of the sources of parallelism in our CPU benchmarks and the challenges in exploiting them.

Overview: Table II presents an overview of different benchmarks and their corresponding parallelism motifs based on the taxonomy provided in [45]. `bsw`, `phmm`, `chain`, `spoa` and `abea` are dynamic programming based but have important differences. The key ones are: (1) type of data dependency present (e.g., 1D / 2D), (2) amount of computation needed (e.g., banded/full matrix), (3) type of matrix traversal (e.g., wavefront/row-wise) and (4) type of input (e.g., sequence/graph). Also present in the benchmark suite are kernels that manipulate hash tables and perform graph construction (`dbg`, `spoa`).

Some of the GenomicsBench benchmarks like `grm` and `kmer-cnt` have **regular compute** patterns since their inputs come in regular, pre-determined sizes. In contrast, a majority of the GenomicsBench benchmarks work on inputs with varying sizes and characteristics and have **irregular compute** patterns. Table III shows the data-parallellism granularity for each of the irregular compute GenomicsBench benchmarks and the corresponding data-parallel computation performed. Note that it is possible to reduce the data-parallelism granularity further by vectorizing each of the data-parallel computations shown in the second column of Table III. However, this comes with significant additional complexity arising from the complex data dependencies present in the benchmarks (Figure 2). To overcome this, implementations often speculate on the absence of data dependencies to achieve high performance (e.g., [70]).

This complexity can often be traded-off for abundant parallelism to be exploited across two other dimensions: (1) read-level parallelism and (2) genome region-level parallelism as shown in Table III. Since each of the benchmarks process millions to billions of reads across millions of genome regions there exists abundant data-parallelism across both these dimensions. Several software tools have adopted this approach. For example, BWA-MEM2 [25] has demonstrated significant benefits by vectorizing inter-sequence computation instead of vectorizing the cell updates for `bsw`.

Challenges in exploiting data parallelism: In spite of abundant read-level / genome region-level parallelism in the GenomicsBench benchmarks, it is difficult to exploit them effectively in different software tools. To understand why, consider the following hypothetical scenario where each data-parallel computation entity discussed in Table III is assigned to a separate vector lane. Each vector is replaced with a new batch of tasks as soon as all of the ones currently

Benchmark	Input Datatype	Applications	Chosen Tool	% Time Spent in Tool (single-thread)	Parallelism Motif
fmi	Short reads	Read Alignment Metagenomics Classification	BWA-MEM2	38%	Tree Traversal
bsw	Short reads	Read Alignment De-Novo Assembly	BWA-MEM2	31%	Dynamic Programming
dbg	Short reads	Variant Calling De-Novo Assembly	Platypus	65%	Graph Construction Hash Table
phmm	Short reads	Variant Calling Error Correction	GATK Haplotype Caller	70%	Dynamic Programming
chain	Long reads	De-Novo Assembly Read Alignment	Minimap2	47.4 %	Dynamic Programming (1D)
spoa	Long reads	Error Correction	Racon	75 %	Dynamic Programming Graph Construction
abea	Long reads	Basecalling Variant Calling	Nanopolish	71.4%	Dynamic Programming
grm	NA	Population Genomics	PLINK2	92.8 %	Dense Matrix Multiplication
nn-base	Long reads	Basecalling	Bonito	95 %	FP Matrix Multiplication
nn-variant	Long reads	Variant Calling	Clair	57.2 %	FP Matrix Multiplication
kmer-cnt	Long reads	De-Novo Assembly	Flye	10%	Hash Table
pileup	Long reads	Variant Calling	Medaka	6.3 % [1]	—

TABLE II

CATEGORIZATION OF BENCHMARKS. FOR BENCHMARKS WITH UTILITY IN MORE THAN ONE APPLICATION, THE SELECTED APPLICATION IS UNDERLINED.

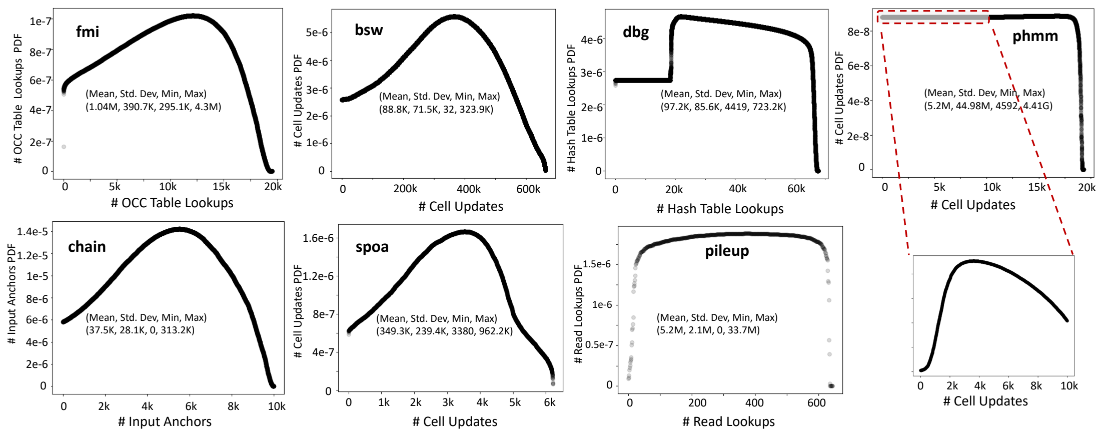

Fig. 4. Distribution of the amount of data-parallel computation performed for each task (x-axis) and its frequency (y-axis) for the different benchmarks. Variations in the computation needed for each task based on its size and data characteristics makes it difficult to exploit the abundant parallelism present in each benchmark. To enable comparison across benchmarks, the normal probability distribution function (PDF) has been used to represent the frequency of computation on the y-axis.

Benchmark	Parallelism Granularity	Data-Parallel Computation
fmi	Read batch	# OCC Table Lookups
bsw	Seed	# Cell Updates
dbg	Genome Region	# Hash Table Lookups
phmm	Genome Region	# Cell Updates
chain	Read	# Input Anchors
spoa	Read Chunk Window	# Cell Updates
pileup	Genome Region	# Read Lookups

TABLE III

PARALLELISM GRANULARITY AND DATA-PARALLEL COMPUTATION FOR IRREGULAR CPU BENCHMARKS. OTHER REGULAR COMPUTE BENCHMARKS NOT SHOWN

assigned to it complete. For vectorization to be efficient, all the tasks assigned to each lane must perform a similar amount of computation. Any imbalances in the computation across vector lanes can severely reduce the efficiency of vector computation and lead to control divergence. For this reason, the inputs to the `bsw` kernel, for example, are sorted based on sequence lengths before being assigned to SIMD lanes. However, even if input sequence lengths have been accounted for, differences in input sequence content can greatly influence the computation performed in each SIMD lane. This is because matrix computation can also be aborted early when aligning highly dissimilar sequences of similar length using `bsw`. As a result we find that the AVX2 16-bit inter-sequence vectorized `bsw` implementation in GenomicsBench performs 2.2× more cell updates than the scalar implementation. Note that the vectorization challenges outlined above exist not only for CPU-based software tools but also for GPUs, which also employ SIMD units to increase compute density.

Similar observations can also be made for the other irregular CPU compute benchmarks. It can be seen from Figure 4 that there exists significant variation in the amount of data-parallel computation performed by different tasks in different benchmarks. For `phmm`, which computes the most likely haplotype given supporting reads, certain genome regions can have up to 1000× imbalance in the computation needed when compared to the average case (as can be seen from the mean ($5.2M$) and maximum ($4.41G$) cell update values across different regions). However, it must be noted that regions with such low or high computational demand are fewer (as indicated by the lightly shaded circles). Across different benchmarks we find that the ratios of maximum to average computation per task can vary from 4.1× to 8.3×.

2) *GPU benchmarks*

Whereas the CPU kernels selected for this benchmark suite were diverse and often encountered challenges in exploiting data parallelism, the GPU kernels we investigated had fairly regular control flow and compute patterns. Predictable control flow and data accesses are a prerequisite for efficient utilization of GPU computing resources, and `abea` and `nn-base` were likely implemented on the GPU for this reason.

	abea	nn-base
Branch efficiency	100 %	100 %
Warp efficiency	75.09 %	100 %
Non-predicated warp efficiency	70.18 %	94.43 %
SM utilization	70.53 %	99.83 %
Occupancy	31.41 %	88.47 %

TABLE IV
GPU KERNEL CONTROL FLOW AND COMPUTE REGULARITY

The `abea` and `nn-base` kernels both avoid branch divergence entirely, and achieve relatively high warp efficiency. This is shown in Table IV. Warp efficiency is defined as the average fraction of active threads in a warp, and "non-predicated" efficiency restricts the definition of active to threads which are not executing predicated instructions. Neural network basecallers such as Bonito break sequences of raw nanopore signal into regular chunks of 4,000 consecutive measurements and feed that data into a fixed-size neural network. Since floating point matrix multiplication is computationally intensive and involves very little control flow, `nn-base` is able to achieve perfect warp efficiency and nearly-complete occupancy and SM utilization. The few predicated instructions reducing overall throughput are likely due to the fact that the neural network of `nn-base` does not operate using filters of sizes which are integer multiples of 32, the number of threads in a warp. On the other hand, the `abea` kernel performs a dynamic programming matrix computation instead of matrix multiplication, it is limited by the execution and memory dependencies inherent to the structure of the computation. Furthermore, `abea` requires frequent synchronization between warps. As a result, the SM utilization and warp efficiency are lower.

C. Instruction Diversity

Instruction diversity characterization helps determine the complexity of functional units needed for specialized hardware. Figure 5 shows the dynamic instruction breakdown for the different CPU benchmarks. The "Other" category includes string, system call, prefetching, and synchronization instructions.

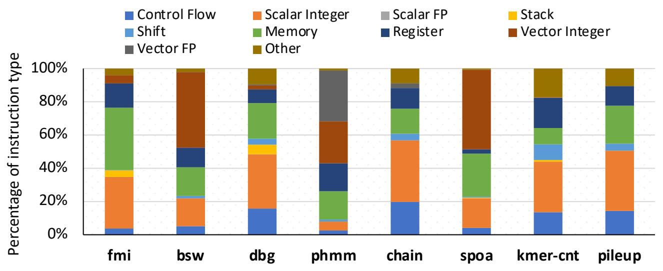

Fig. 5. Breakdown of dynamic instructions in different benchmarks. `grm` is excluded because its multithreaded design to decompress inputs affects the accuracy of measurements from the MICA pintool.

Among the benchmarks analyzed, `phmm`, `bsw`, and `spoa` benefit from SIMD vectorization and have a high proportion of vector computation instructions. It can be also be seen that `phmm` is the only CPU kernel that performs floating point computation, while the other kernels are dominated by scalar integer computation. `phmm` also uses single-precision floating point computation in most cases, and resorts to double-precision floating point only in rare cases when single-precision is insufficient to represent the result. `bsw`, `phmm`, and `chain` are compute-intensive and have a lower proportion of memory loads and stores when compared to memory-intensive benchmarks like `fmi`. We also looked at the common operations performed in vectorized benchmarks. For instance, `bsw` uses `blend` instructions for cell updates and band adjustment, and `spoa` extensively uses `shift` instructions to compare against cells present in a previous column or diagonal but which are part of a different SIMD vector.

D. Memory Access Characteristics

1) CPU benchmarks

In this section, we perform a detailed characterization of the memory access patterns of different GenomicsBench benchmarks.

Off-chip Data Requirements: Figure 6 shows the off-chip

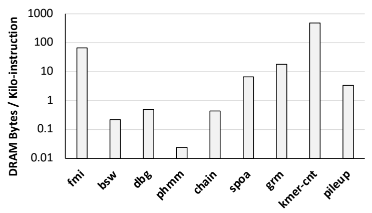

Fig. 6. Off-chip data requirements for different benchmarks.

data requirements for different GenomicsBench benchmarks. It can be seen that benchmarks like `fmi` and `kmer-cnt` have significantly higher off-chip data requirements, measured in DRAM bytes per kilo-instruction (BPKI) (66.8 BPKI and 484.1 BPKI respectively). For `fmi` and `kmer-cnt` the memory access bottlenecks are due to irregular memory accesses over large working sets, ~10 GB (FM-index) and ~8 GB (hash table) respectively, with little spatial or temporal locality. In `kmer-cnt`, there is low spatial locality because a 1-2 byte counter is updated for every 64 bytes (cache block) read from memory. Potential approaches to improve `kmer-cnt` performance include implementing cache-friendly hashing techniques like robin hood hashing [71], and improving temporal locality since the k-mers to be inserted into the hash table are known a priori.

In contrast, other benchmarks like `spoa` have modest off-chip data requirements (6.62 BPKI), while compute-intensive benchmarks like `phmm` have much lower data movement (0.02 BPKI) from off-chip memory.

Cache Miss Rates: Figure 8 shows the L1 and L2 cache miss rates and percentage of CPU cycles spent stalling for data. Notably in `fmi` and `kmer-cnt`, 41.5% and 69.2% of CPU cycles are spent waiting for data. While `fmi` uses all the bytes in a cache block when performing OCC table lookups, `kmer-cnt` only updates a 1-2 byte counter per LLC miss

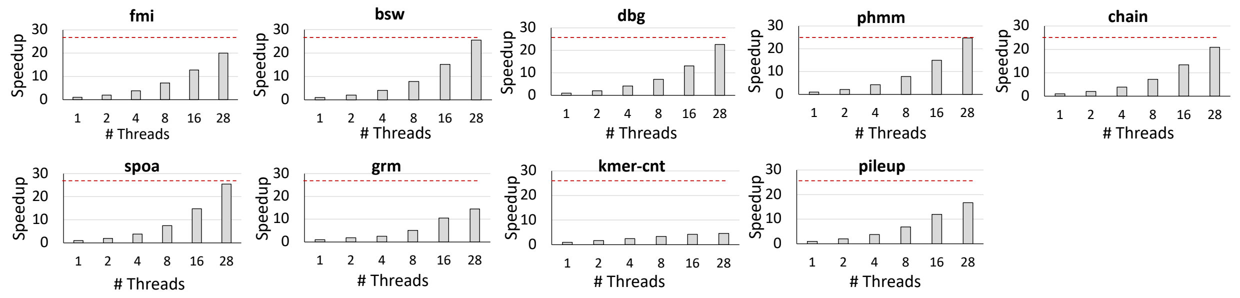

Fig. 7. Thread scaling for different kernels in GenomicsBench. Dotted red line shows the maximum speedup achievable on the experimental system with 28 cores. Experiments were performed on a dual socket (14-core per socket Haswell machine (Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz, AVX2) with 35 MB LLC)

and has poor spatial locality. Apart from these two, other benchmarks spend <20% of CPU cycles waiting for data.

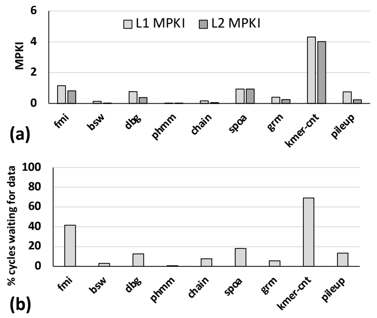

Fig. 8. (a) L1 and L2 misses per kilo-instruction (MPKI) (b) Percentage of CPU cycles spent for waiting for data.

2) GPU benchmarks

When accessing global memory, `abea` and `nn-base` kernels were unable to achieve peak memory bandwidth due to strided or irregular data accesses. This is shown in Table V.

	abea	nn-base
Global Load Efficiency	25.5 %	70.3 %
Global Store Efficiency	68.5 %	100 %

TABLE V
USEFUL PROPORTION OF GPU GLOBAL MEMORY BANDWIDTH USED.

The extent of irregularity of memory accesses in both GPU kernels is a direct artifact of the type of computation performed. For `nn-base`, neural network model weights and inputs can be loaded in several large accesses at the start of computation. Since Bonito's convolutional neural network is comprised of many layers of separable convolutions, these matrix vector multiplications are not too large and can be performed in shared memory. At the end, results are written to global memory in contiguous transactions. For the `abea` kernel, however, there are dependencies between consecutive diagonal bands of the dynamic programming matrix which are computed. In order to calculate the matrix efficiently, the previous three rows (which the following band computation is dependent on) are stored in Shared Memory. This leaves no room to cache the reference's k-mer current model and other frequently accessed data in Shared Memory. The resulting accesses to global memory are performed with sub-optimal efficiency due to the decreased spatial locality of data accesses.

E. Thread Scaling

Figure 7 shows the thread scaling behavior of the multi-threaded versions of the irregular CPU benchmarks. All inputs to these benchmarks are grouped into independent tasks with each task dynamically scheduled on a CPU thread using OpenMP. Almost all GenomicsBench benchmarks benefit from coarse-grained task-level parallelism. It can be seen that most of the benchmarks achieve perfect scaling (`bsw`, `dbg`, `phmm` and `spoa`), while `fmi` and `chain` achieve near-perfect scaling. `kmer-cnt` uses close to the peak random access memory-bandwidth on our system and does not scale well with increasing number of threads, whereas `pileup` suffers from random memory accesses.

F. Microarchitectural Bottleneck Analysis

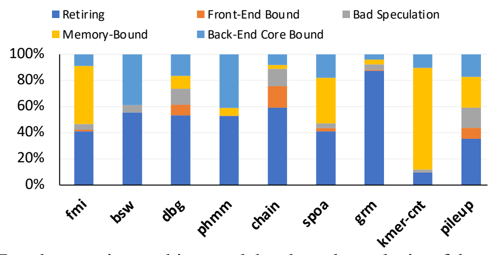

Fig. 9. Top-down microarchitectural bottleneck analysis of kernels (single thread).

Figure 9 shows the results of top-down analysis [72] of performance bottlenecks. It can be seen that memory-bound benchmarks like `fmi` and `kmer-cnt` spend 44.4% and 86.6% of their pipeline slots waiting for data. For `fmi`, >80% of OCC table accesses lead to opening of a new DRAM page making the accesses highly irregular. There is also little spatial or temporal locality in k-mer counting. Each update to the k-mer count table results in a last-level cache miss leading to significant memory-latency related stalls. Some of these stalls could potentially be mitigated by implementing software prefetching [71], since the k-mers to be looked up are known in advance. Compute-intensive benchmarks like `bsw`, `chain` and `phmm` spend >50% of their pipeline slots retiring instructions. They are bottlenecked by backend core resources because of limited number of available ports for scheduling vector and floating point instructions. `grm` performs CPU-friendly dense matrix multiplication and makes best use of available CPU pipeline slots (87.70% retiring). The memory-related stalls in `spoa` and `pileup` result from cache misses during incremental update of the partial-order alignment graph and random accesses to the read alignment records respectively.

V. Related Work

The BioPerf benchmark suite [14] evaluates several DNA and protein sequence analysis benchmarks such as BLAST [73] and HMMER [74] and provides pre-compiled Alpha binaries with Simpoints to facilitate architectural simulation. Some of these benchmarks are further characterized in BioBench [15] and have been shown to have a high ILP. Recent updates to these benchmarks have also been proposed in BioBench2. GenomicsBench improves coverage of these prior benchmark suites by including both vectorized dynamic programming kernels (`bsw`, `phmm`, `spoa`) and GPU-optimized dense neural network kernels (`nn-base`, `nn-variant`). Among the GenomicsBench benchmarks, only `phmm` and `bsw` share similarities with HMMER and BLAST in performing floating point matrix computation and local alignment with Smith-Waterman respectively. Other benchmarking efforts also focus on characterizing the BioBench benchmarks on different architectural platforms [75], [76] and increasing its usability [77]. GenomicsBench leverages some of the optimized implementations for different short-read benchmarks proposed in prior work [71], and expands its scope to cover a broader diversity of sequence analysis steps and includes long-read benchmarks. Prior work [78] identifies some of the key computation kernels in secondary analysis, but with a focus on short reads. Intel's Genomics Kernel Library (GKL) [79] also includes reference vectorized implementations of a few short-read benchmarks like `phmm`.

VI. Conclusion

In this paper, we present the GenomicsBench benchmark suite, containing 12 computationally intensive genomics kernels drawn from popular bioinformatics software tools. We perform detailed instruction level and microarchitectural analysis on these kernels to expose their performance bottlenecks. We also observe that the irregular data-parallelism in these benchmarks cannot be easily exploited by commodity hardware. GenomicsBench will be open sourced to the broader research community.

Acknowledgment

We thank the anonymous reviewers for their valuable feedback and suggestions which helped improve this paper.

References

[1] M. van Lanschot, L. Bosch, M. de Wit, B. Carvalho, and G. Meijer, "Early detection: The impact of genomics," *Virchows Archiv*, vol. 471, no. 2, pp. 165–173, 2017.

[2] C. H. June, R. S. O'Connor, O. U. Kawalekar, S. Ghassemi, and M. C. Milone, "Car t cell immunotherapy for human cancer," *Science*, vol. 359, no. 6382, pp. 1361–1365, 2018.

[3] E. Zeggini, A. L. Gloyn, A. C. Barton, and L. V. Wain, "Translational genomics and precision medicine: Moving from the lab to the clinic," *Science*, vol. 365, no. 6460, pp. 1409–1413, 2019.

[4] "Personalised medicine at fda. a progress and outlook report," http://www.personalizedmedicinecoalition.org/Userfiles/PMC-Corporate/file/PM_at_FDA_A_Progress_and_Outlook_Report.pdf.

[5] "Artic network. real-time molecular epidemiology for outbreak response," https://artic.network/.

[6] "Next-generation-sequencing.v1.5.7," https://twitter.com/AlbertVilella/status/1291669705799983106.

[7] "Mgi delivers the $100 genome at agbt conference," https://www.genengnews.com/news/mgi-delivers-the-100-genome-at-agbt-conference/.

[8] "Mgi unveils extreme throughput sequencing platform at agbt to enable $100 human genome," https://www.genomeweb.com/sequencing/mgi-unveils-extreme-throughput-sequencing-platform-agbt-enable-100-human-genome.

[9] "Novaseq 6000," https://sapac.illumina.com/content/dam/illumina-marketing/documents/products/datasheets/novaseq-6000-system-specification-sheet-770-2016-025.pdf.

[10] S. Kovaka, Y. Fan, B. Ni, W. Timp, and M. C. Schatz, "Targeted nanopore sequencing by real-time mapping of raw electrical signal with uncalled," *BioRxiv*, 2020.

[11] "The all of us research program," https://allofus.nih.gov/.

[12] "Uk plans to sequence 5 million genomes in 5 years," https://www.bionews.org.uk/page_138891.

[13] "India to launch its 1st human genome cataloguing project."

[14] D. A. Bader, Y. Li, T. Li, and V. Sachdeva, "Bioperf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications," in *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.* IEEE, 2005, pp. 163–173.

[15] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng, and D. Yeung, "Biobench: A benchmark suite of bioinformatics applications," in *IEEE International Symposium on Performance Analysis of Systems and Software, 2005. ISPASS 2005.* IEEE, 2005, pp. 2–9.

[16] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "Minebench: A benchmark suite for data mining workloads," in *2006 IEEE International Symposium on Workload Characterization*. IEEE, 2006, pp. 182–188.

[17] "Microsoft unveils genomics innovation and new partners at ashg 2018," https://cloudblogs.microsoft.com/industry-blog/health/2018/10/15/microsoft-unveils-genomics-innovation-and-new-partners-at-ashg-2018/.

[18] A. Cacho, E. Smirnova, S. Huzurbazar, and X. Cui, "A comparison of base-calling algorithms for illumina sequencing technology," *Briefings in bioinformatics*, vol. 17, no. 5, pp. 786–795, 2016.

[19] R. R. Wick, L. M. Judd, and K. E. Holt, "Performance of neural network basecalling tools for oxford nanopore sequencing," *Genome biology*, vol. 20, no. 1, p. 129, 2019.

[20] "Bonito," https://github.com/nanoporetech/bonito.

[21] "Nanopore sequencing accuracy," https://nanoporetech.com/accuracy.

[22] J. L. Weirather, M. de Cesare, Y. Wang, P. Piazza, V. Sebastiano, X.-J. Wang, D. Buck, and K. F. Au, "Comprehensive comparison of pacific biosciences and oxford nanopore technologies and their applications to transcriptome analysis," *F1000Research*, vol. 6, 2017.

[23] K. Shafin, T. Pesout, R. Lorig-Roach, M. Haukness, H. E. Olsen, C. Bosworth, J. Armstrong, K. Tigyi, N. Maurer, S. Koren *et al.*, "Nanopore sequencing and the shasta toolkit enable efficient de novo assembly of eleven human genomes," *Nature Biotechnology*, pp. 1–10, 2020.

[24] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.

[25] V. Md, S. Misra, H. Li, and S. Aluru, "Efficient architecture-aware acceleration of bwa-mem for multicore systems," in *Proceedings of the Thirty-Third IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2019.

[26] "Genome analysis toolkit: Variant discovery in high-throughput sequencing data," https://software.broadinstitute.org/gatk/.

[27] "Germline short variant discovery (snps + indels). best practices workflow," https://software.broadinstitute.org/gatk/best-practices/workflow?id=11145.

[28] M. Vasimuddin, S. Misra, and S. Aluru, "Identification of significant computational building blocks through comprehensive investigation of NGS secondary analysis methods," *[Preprint] bioRXiv*, April 2018.

[29] M. Mahmoud, N. Gobet, D. I. Cruz-Dávalos, N. Mounier, C. Dessimoz, and F. J. Sedlazeck, "Structural variant calling: the long and the short of it," *Genome biology*, vol. 20, no. 1, p. 246, 2019.

[30] E. E. Schadt, S. Turner, and A. Kasarskis, "A window into third-generation sequencing," *Human molecular genetics*, vol. 19, no. R2, pp. R227–R240, 2010.

[31] M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner, "Assembly of long, error-prone reads using repeat graphs," *Nature biotechnology*, vol. 37, no. 5, pp. 540–546, 2019.

[32] R. Vaser, I. Sović, N. Nagarajan, and M. Šikić, "Fast and accurate de novo genome assembly from long uncorrected reads," *Genome research*, vol. 27, no. 5, pp. 737–746, 2017.
[33] "Oxford nanopore minion." https://nanoporetech.com/products/minion.
[34] J. Quick, N. D. Grubaugh, S. T. Pullan, I. M. Claro, A. D. Smith, K. Gangavarapu, G. Oliveira, R. Robles-Sikisaka, T. F. Rogers, N. A. Beutler *et al.*, "Multiplex pcr method for minion and illumina sequencing of zika and other virus genomes directly from clinical samples," *Nature protocols*, vol. 12, no. 6, p. 1261, 2017.
[35] C. Li, K. R. Chng, E. J. H. Boey, A. H. Q. Ng, A. Wilm, and N. Nagarajan, "Inc-seq: accurate single molecule reads using nanopore sequencing," *GigaScience*, vol. 5, no. 1, pp. s13 742–016, 2016.
[36] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.
[37] D. Kim, L. Song, F. P. Breitwieser, and S. L. Salzberg, "Centrifuge: rapid and sensitive classification of metagenomic sequences," *Genome research*, vol. 26, no. 12, pp. 1721–1729, 2016.
[38] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature methods*, vol. 9, no. 4, p. 357, 2012.
[39] R. Luo, T. Wong, J. Zhu, C.-M. Liu, X. Zhu, E. Wu, L.-K. Lee, H. Lin, W. Zhu, D. W. Cheung *et al.*, "Soap3-dp: fast, accurate and sensitive gpu-based short read aligner," *PloS one*, vol. 8, no. 5, p. e65632, 2013.
[40] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.
[41] E. W. Myers and W. Miller, "Optimal alignments in linear space," *Bioinformatics*, vol. 4, no. 1, pp. 11–17, 1988.
[42] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy, "Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation," *Genome research*, vol. 27, no. 5, pp. 722–736, 2017.
[43] P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner, "Error correction of high-throughput sequencing datasets with non-uniform coverage," *Bioinformatics*, vol. 27, no. 13, pp. i137–i141, 2011.
[44] R. Ounit, S. Wanamaker, T. J. Close, and S. Lonardi, "Clark: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers," *BMC genomics*, vol. 16, no. 1, p. 236, 2015.
[45] K. Yelick, A. Buluç, M. Awan, A. Azad, B. Brock, R. Egan, S. Ekanayake, M. Ellis, E. Georganas, G. Guidi *et al.*, "The parallelism motifs of genomic data analysis," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190394, 2020.
[46] "E.coli ont data. loman lab," https://zenodo.org/record/1172816/files/Loman_E.coli_MAP006-1_2D_50x.fasta.
[47] A. Rimmer, H. Phan, I. Mathieson, Z. Iqbal, S. R. Twigg, A. O. Wilkie, G. McVean, and G. Lunter, "Integrating mapping-, assembly-and haplotype-based approaches for calling variants in clinical sequencing applications," *Nature genetics*, vol. 46, no. 8, pp. 912–918, 2014.
[48] M. A. Eberle, E. Fritzilas, P. Krusche, M. Källberg, B. L. Moore, M. A. Bekritsky, Z. Iqbal, H.-Y. Chuang, S. J. Humphray, A. L. Halpern *et al.*, "A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree," *Genome research*, vol. 27, no. 1, pp. 157–164, 2017.
[49] R. Poplin, V. Ruano-Rubio, M. A. DePristo, T. J. Fennell, M. O. Carneiro, G. A. Van der Auwera, D. E. Kling, L. D. Gauthier, A. Levy-Moonshine, D. Roazen *et al.*, "Scaling accurate genetic variant discovery to tens of thousands of samples," *BioRxiv*, p. 201178, 2017.
[50] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between fpga and gpu," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 127–135.
[51] "Caenorhabditis elegans 40x coverage dataset, pacific biosciences." http://datasets.pacb.com.s3.amazonaws.com/2014/c_elegans/list.html.
[52] R. Vaser, I. Sović, N. Nagarajan, and M. Šikić, "Fast and accurate de novo genome assembly from long uncorrected reads," *Genome research*, vol. 27, no. 5, pp. 737–746, 2017.
[53] C. Lee, C. Grasso, and M. F. Sharlow, "Multiple sequence alignment using partial order graphs," *Bioinformatics*, vol. 18, no. 3, pp. 452–464, 2002.
[54] C. Lee, "Generating consensus sequences from partial order multiple sequence alignment graphs," *Bioinformatics*, vol. 19, no. 8, pp. 999–1008, 2003.
[55] H. Gamaarachchi, C. W. Lam, G. Jayatilaka, H. Samarakoon, J. T. Simpson, M. A. Smith, and S. Parameswaran, "Gpu accelerated adaptive banded event alignment for rapid comparative nanopore signal analysis," *BMC bioinformatics*, vol. 21, no. 1, pp. 1–13, 2020.
[56] N. J. Loman, J. Quick, and J. T. Simpson, "A complete bacterial genome assembled de novo using only nanopore sequencing data," *Nature methods*, vol. 12, no. 8, pp. 733–735, 2015.
[57] H. Suzuki and M. Kasahara, "Introducing difference recurrence relations for faster semi-global alignment of long sequences," *BMC bioinformatics*, vol. 19, no. 1, pp. 33–47, 2018.
[58] "Nanopore wgs consortium data," https://github.com/nanopore-wgs-consortium/NA12878/blob/master/Genome.md.
[59] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes *et al.*, "Nanopore sequencing and assembly of a human genome with ultra-long reads," *Nature biotechnology*, vol. 36, no. 4, pp. 338–345, 2018.
[60] C. C. Chang, C. C. Chow, L. C. Tellier, S. Vattikuti, S. M. Purcell, and J. J. Lee, "Second-generation plink: rising to the challenge of larger and richer datasets," *Gigascience*, vol. 4, no. 1, pp. s13 742–015, 2015.
[61] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.
[62] "Medaka," https://github.com/nanoporetech/medaka.
[63] "Sequence alignment map format specification," https://samtools.github.io/hts-specs/SAMv1.pdf.
[64] "Oxford nanopore variant calling workflow (limited support release)," https://github.com/kishwarshafin/pepper/blob/r0.1/docs/PEPPER_variant_calling.md.
[65] R. Luo, C.-L. Wong, Y.-S. Wong, C.-I. Tang, C.-M. Liu, C.-M. Leung, and T.-W. Lam, "Clair: exploring the limit of using a deep neural network on pileup data for germline variant calling," *Nature Machine Intelligence*, vol. 2, no. 4, pp. 220–227, 2020.
[66] "Intel vtune profiler," https://software.intel.com/content/www/us/en/develop/tools/vtune-profiler.html.
[67] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *IEEE micro*, vol. 27, no. 3, pp. 63–72, 2007.
[68] "runsa/runss custom command line analysis," https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/command-line-interface/running-command-line-analysis/running-runsa-runss-custom-analysis-from-the-command-line.html.
[69] "Nvidia visual profiler," https://developer.nvidia.com/nvidia-visual-profiler.
[70] M. Farrar, "Striped smith–waterman speeds database searches six times over other simd implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.
[71] S. Misra, T. C. Pan, K. Mahadik, G. Powley, P. N. Vaidya, M. Vasimuddin, and S. Aluru, "Performance extraction and suitability analysis of multi-and many-core architectures for next generation sequencing secondary analysis," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, 2018, pp. 1–14.
[72] A. Yasin, "A top-down method for performance analysis and counters architecture," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014, pp. 35–44.
[73] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.
[74] S. R. Eddy, "Profile hidden markov models." *Bioinformatics (Oxford, England)*, vol. 14, no. 9, pp. 755–763, 1998.
[75] W. Chen and C. Hong, "Pbb: a parallel bioinformatics benchmark suite for shared memory multiprocessors," in *Proceedings of the 2007 Asian technology information program's (ATIP's) 3rd workshop on High performance computing in China: solution approaches to impediments for high performance computing*. ACM, 2007, pp. 141–144.
[76] V. Sachdeva, E. Speight, M. Stephenson, and L. Chen, "Characterizing and improving the performance of bioinformatics workloads on the power5 architecture," in *2007 IEEE 10th International Symposium on Workload Characterization*. IEEE, 2007, pp. 89–97.
[77] M. Hanussek, F. Bartusch, and J. Krüger, "Bootable: Bioinformatics benchmark tool suite for applications and hardware," *Future Generation Computer Systems*, vol. 102, pp. 1016–1026, 2020.
[78] M. Vasimuddin, S. Misra, and S. Aluru, "Identification of significant computational building blocks through comprehensive deep dive of ngs secondary analysis methods," *bioRxiv*, p. 301903, 2018.
[79] P. Foley, A. Prabhakaran, K. Gururaj, M. Naik, S. Gopalan, A. Shargorodskiy, and E. Brau, "Accelerate genomics research with the broad-intel genomics stack," 2017.