MaxSSmap: A GPU program for mapping divergent short reads to genomes with the maximum scoring subsequence

Turki Turki∗ Usman Roshan†‡

Abstract

Programs based on hash tables and Burrows-Wheeler are very fast for mapping short reads to genomes but have low accuracy in the presence of mismatches and gaps. Such reads can be aligned accurately with the Smith-Waterman algorithm but it can take hours and days to map millions of reads even for bacteria genomes. We introduce a GPU program called MaxSSmap with the aim of achieving comparable accuracy to Smith-Waterman but with faster runtimes. Similar to mainstream approaches MaxSSmap identifies a local region of the genome followed by exact alignment. Instead of using hash tables or Burrows-Wheeler in the first part, MaxSSmap calculates maximum scoring subsequence score between the read and disjoint fragments of the genome in parallel on a GPU and selects the highest scoring fragment for exact alignment. We evaluate MaxSSmap’s accuracy and runtime when mapping simulated E.coli and human reads of 10% to 30% mismatches with gaps of various lengths to the E.coli genome and human chromosome one respectively. We show that MaxSSmap attains comparable high accuracy and low error to fast Smith-Waterman programs yet has much lower runtimes. We also show that MaxSSmap can map reads rejected by fast mappers with high accuracy and low error much faster than if Smith-Waterman were used. The MaxSSmap source code is freely available from http://www.cs.njit.edu/usman/MaxSSmap.

1 Introduction

In next generation sequencing experiments we may encounter divergent reads in various scenarios. These include structural variation studies, comparison of distantly related genomes, absence of same species reference genome, sequence error in long reads, genome variation within same species, and mRNA-seq experiments [1][2][3][4][5][6][7][8][9]. Mainstream programs [10][11] based on hash tables and Burrows-Wheeler tranform are very fast but have low accuracy on such reads that tend to

∗Computer Science Department, King Abdulaziz University, P.O. Box 80221, Jeddah 21589, Saudi Arabia
†To whom correspondence should be addressed; usman@cs.njit.edu, Ph: 862-485-0590, Fax: 973-596-5777
‡Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102
contain many mismatches and gaps [1, 12]. The Smith-Waterman algorithm [13] can map divergent reads accurately but is considerably expensive. Even high performance multi-core and Graphics Processing Unit (GPU) implementations can take hours and days to align millions of reads even to bacteria genomes. As a solution we introduce a GPU program called MaxSSmap with the aim of achieving comparable accuracy to Smith-Waterman on divergent reads but with faster runtimes.

We divide the genome into same size disjoint fragments and then map a read to all fragments in parallel on a GPU with the maximum scoring subsequence score [14, 15]. A GPU can run several hundred threads at the same time and allows for massive parallelism in computer programs (see http://www.gpucomputing.net). The maximum scoring subsequence is roughly the same as Smith-Waterman except that it does not consider gaps. Once we identify the first and second highest scoring fragments — we need the second to eliminate repeats — we perform Needleman-Wunsch alignment of the read to the identified region of genome. We present a GPU program called MaxSSmap that implements this idea along with several heuristics and shortcuts that lead to faster runtimes without sacrificing accuracy.

On reads with fewer than 10% mismatches our program offers no advantage over mainstream approaches. Programs like NextGenMap that use hash-tables in their first phase can map such reads very quickly with high accuracy compared to other leading programs [12]. Thus we focus on reads with divergence between 10% and 30% as well as gaps of lengths up to 30 both in the read and genome.

We compare MaxSSmap to two fast Smith-Waterman programs. The first is the recently published Smith-Waterman library for short read mapping and protein database search called SSW [16]. This uses a fast Single-Instruction-Multiple-Data Smith-Waterman algorithm to align a given read to the entire genome. The authors of the program demonstrate improved and comparable runtimes to state of the art fast Smith-Waterman programs for mapping DNA sequences to a genome. In addition this produces output in SAM format and has also been applied it to real data in the context of realigning unmapped reads [16]. The second is a fast GPU Smith-Waterman program for protein sequence database search called CUDA-SW++ [17]. We note that this is not designed for mapping DNA sequence reads. However, we adapt it to short read mapping by considering fragments of the genome as database sequences and read as the query.

Exact Smith-Waterman methods take much longer than hash-table and Burrows-Wheeler based programs to align millions of reads to genome sequences. In this setting we study several meta-methods that first align reads with a fast program and then map rejected ones with a slower but more accurate one such as MaxSSmap and SSW.

We study accuracy and runtime for mapping simulated E.coli and human reads of various lengths to the E.coli and human chromosome one. Our focus is on reads with 10% to 30% mismatches and gaps up to length 30. We show that MaxSSmap attains comparable high accuracy and low error as CUDA-SW++ and SSW but is at least 8 and 60 times faster than the two programs respectively. We also show that MaxSSmap can map reads rejected by NextGenMap [12] with high accuracy and low error and much faster than if Smith-Waterman were used.

Below we provide basic background and describe our program in detail. We then present an experimental performance study on simulated E.coli and human chromosome one reads of various divergence.
2 Methods

2.1 Background

Before we describe MaxSSmap we provide background on the maximum scoring subsequence and GPUs and CUDA.

2.1.1 Maximum scoring subsequence

The maximum scoring subsequence for a sequence of real numbers \( \{x_1, x_2, \ldots, x_n\} \) is defined to be the contiguous subsequence \( \{x_i, \ldots, x_j\} \) that maximizes the sum \( x_i + \ldots + x_j \) \( (0 \leq i, j \leq n) \). A simple linear time approach will find the maximum scoring subsequence \([14, 15]\). To apply this to DNA sequences consider two of the same length aligned to each other without gaps. Each aligned character corresponds to a substitution whose cost can be obtained from a position specific scoring matrix that accounts for base call probabilities, or a substitution scoring matrix, or a trivial match or mismatch cost. The maximum scoring subsequence between the two DNA sequences can now be obtained through this sequence of substitution scores \([14, 15]\).

2.1.2 Graphics Processing Units (GPUs) and CUDA

CUDA is a programming language that is developed by NVIDIA. It is mainly C with extensions for programming on NVIDIA GPUs. We use CUDA version 4.2 for all GPU programs in this study. The GPU is designed for running in parallel hundreds of short functions called threads. Threads are organized into blocks which in turn are organized into grids. We use one grid and automatically set the number of blocks to the total number of genome fragments divided by the number of threads to run in a block. The number of threads in a block can be specified by the user and otherwise we set it to 256 by default.

The GPU memory is of several types each with different size and access times: global, local, constant, shared, and texture. Global memory is the largest and can be as much as 6GB for Tesla GPUs. Local memory is the same as global memory but limited to a thread. Access times for global and local memory are much higher than the those for a CPU program to access RAM. However, this time can be considerably reduced with coalescent memory access that we explain below. Constant and texture are cached global memory and accessible by any thread in the program. Shared is on-chip making it the fastest and is limited to threads in a block. More details about CUDA and the GPU architecture can be found in the NVIDIA online documentation \([18]\) and recent books \([19, 20]\).

2.2 MaxSSmap algorithm

2.2.1 Overview

Our program, that we call MaxSSmap, follows the same two part approach of mainstream mappers: first identify a local region of the genome and then align the read with Needleman-Wunsch (or
Smith-Waterman) to the identified region. The second part is the same as mainstream methods but in the first part we use the maximum scoring subsequence as described below.

MaxSSmap divides the genome into fragments of a fixed size given by the user. It uses one grid and automatically sets the number of blocks to the total number of genome fragments divided by the number of threads to run in a block. The number of threads in a block can be specified by the user and otherwise we set it to 256 by default.

**First phase of MaxSSmap** Each thread of the GPU computes the maximum scoring subsequence of the read and a unique fragment with a sliding window approach (see Figure 1). In order to map across junctions between fragments each thread also considers neighboring fragments when mapping the read (see Figure 2). When done it outputs the fragment number with the highest and second highest score and considers the read to be mapped if the ratio of the second best to best score is below .9 (chosen by empirical performance). This reduces false positives due to repeats.

| Genome | A A C A A C C A G G A G G C C |
|------|-------------------------------|
| Read | T T C C C C A T C             |
| Cost | -4-4 5 5-4 5-4-4-4            |

**Figure 1:** MaxSSmap slides the read against a given fragment of the genome, keeps track of the maximum scoring subsequence in each window, and outputs the position and score of the best one in the end. In the $k^{th}$ window the first nucleotide of the read is mapped to the $k^{th}$ nucleotide of the genome. In the above example the read is shown as mapped to the genome in window four. Each match has cost 5 and a mismatch has cost -4. The shaded region shows the maximum scoring subsequence of score 11.

| Start of mapping | End of mapping |
|------------------|---------------|
| 0 1 2 3          | 0 1 2 3       |
| read             |               |

**Figure 2:** In this figure the genome is divided into four fragments which means four threads will run on the GPU. Thread with ID 1 maps the read to fragment 1 and part of it also maps to nucleotides from fragment 2. Thread 1 slides the read across fragment 1 as described in Figure 1 and stops when it has covered all of fragment 1. Note that we account for junctions between fragments and ensure that the read is fully mapped to the genome.

**Second phase of MaxSSmap** After the fragment number is identified we consider the region of the genome starting from the identified fragment and spanning fragments to the right until we have enough nucleotides as the read sequence. In the second part we align the read sequence with Needleman-Wunsch to the genome region from the first part. The default settings for match, mismatch, and gap costs that we also use in this study are set to 5, -4, and -26.
Incorporating base qualities and position specific scoring matrix  We also consider the base qualities of reads in both phases of the program. This can be done easily by creating a position specific scoring matrix for each read that also allows for fast access using table lookup [17]. For example let \( x \) be the probability that the base at position \( i \) is correctly sequenced. This can be calculated by the phred quality score [21] that is provided with the reads. The score of a match against the nucleotide at position \( i \) is \( \text{match} \times x \) and mismatch is \( \text{mismatch} \times \frac{x}{3} \).

Read lengths, SAM output, and source code  MaxSSmap can also map reads of various lengths present in one fastq file. There is no need to specify the read length. However, the maximum read length is limited to 2432 base pairs (bp) in the current implementation (see paragraph on shared memory below). MaxSSmap outputs in SAM format that is widely used for mapping DNA reads to genomes [22]. The source code is freely available from [http://www.cs.njit.edu/usman/MaxSSmap](http://www.cs.njit.edu/usman/MaxSSmap).

We implement several novel heuristics and take advantage of the GPU architecture to speed up our program which we describe below.

2.2.2 GPU specific heuristics

Coalescent global memory access  Coalesced memory access is a key performance consideration when programming in CUDA (see the CUDA C Best Practices Guide [23]). Roughly speaking, each thread of a GPU has its own unique identifier that we call \( \text{thread\_id} \). In order to have coalescent memory access our program must have threads with consecutive identifiers access consecutive locations in memory (roughly speaking). We achieve this by first considering the genome sequence as rows of fragments of a fixed size. We then transpose this matrix to yield a transposed genome sequence that allows coalescent memory access. The transposed genome is transferred just once in the beginning of the program from CPU RAM to GPU global memory. It has negligible overhead time compared to the total one for mapping thousands of reads. See Figure 3 for a toy genome ACCGTAGGACCA and fragment length of three. If the genome is not a multiple of the fragment length we pad the last fragment with N’s. Our CUDA program runs a total of \( \text{numfragments} \) threads. In the example shown in Figure 3 there are four fragments. Thus our CUDA program would run four threads simultaneously with identifiers zero through three. Each thread would access the transposed genome sequence first at location \( \text{thread\_id} \), then at \( \text{thread\_id} + \text{numfragments} \), followed by location \( \text{thread\_id} + 2 \times \text{numfragments} \), and so on.

Byte packing for faster global memory access  In the GPU we store the genome sequence in a single array of int4 type instead of char. This leads to fewer global memory accesses and thus faster runtimes. To enable this we append ‘N’ characters onto the genome and query until both lengths are multiples of 16. This also requires that the fragment length be a multiple of 16.

Look ahead strategy to reduce global memory penalties  As mentioned earlier MaxSSmap uses a sliding window approach from left to right to map a read to a given fragment on the genome.
Figure 3: Genome sequence in transpose format to enable coalescent memory access. In MaxSSmap threads with IDs 0 through 3 would at the same time read characters A, G, G, and C of the transposed genome to compare against the read. Since the four characters are in consecutive memory locations and so are the thread IDs, our program makes just one read from global memory instead of four separate ones.

In its implementation we compute the score of the read in the current window and sixteen windows to the right at the same time. Therefore instead of shifting the window by one nucleotide we shift it by sixteen. This leads to fewer global memory calls and also allows us to unroll loops. See file MaxSSMap_shared_int4_fast.cu in the source code for exact implementation.

Shared memory We store the query in shared memory to allow fast access. As mentioned earlier the GPU access time to shared memory is fastest. This, however, imposes a limitation on the read length because shared memory size is much smaller than global memory. The Fermi Tesla M2050 GPUs that we use in this study have a maximum of 49152 bytes shared memory per block. The data structure stores the query in a profile format and so occupies a total of \((\text{readlength} + 16) \times 4 \times 5\) bytes. The 4 accounts for number of bytes in a float, 5 is for bases A, C, G, T, and N, and 16 is for additional space used by the look-ahead strategy and to eliminate if-statements in the code. Thus the maximum allowable DNA read length of the current implementation is 2432 bp (largest multiple of 16 below the cap size of 2441 bp). The query length can be increased at the expense of running time by storing the query in constant memory, which is of size 65536 bytes, or in global memory.

2.2.3 General speedup heuristic: Skipping nucleotides in read sequence while calculating maximum subsequence score

Given that we calculate the score of several windows at the same time with our look ahead strategy, we save considerable time if we consider every other nucleotide in the read sequence when computing the maximum scoring subsequence. This heuristic reduces runtime considerably than if we were to compare all nucleotides in the read sequence. In fact we study a faster version called MaxSSmap_fast in which we consider first consider 4 nucleotides in the read sequence, skip the next 12, and continue this cycle until the read length is reached. This is not a problem in our implementation because the read is padded with N’s to make the size a multiple of 16. When this is used in conjunction with a fast mapper (as a meta-method) it has even lower runtimes yet improves accuracy considerably compared to the fast mapper alone. See files MaxSSMap_shared_int4_fast.cu and MaxSSMap_shared_int4_fast6.cu in the source code for exact implementation.
2.2.4 Parallel multi-threaded CPU implementation of MaxSSmap

We have also implemented a parallel multi-threaded CPU implementation of MaxSSmap with the OpenMP library [24] (OpenMP available from [http://www.openmp.org](http://www.openmp.org)). Each thread maps the given read to a unique fragment of the genome. The number of threads is automatically set to the genome size divided by the specified fragment length. Thus if the fragment length is 4800 then for E.coli (approximately 5 million bp) it runs about 1042 threads on the available CPU cores. This also uses the look ahead strategy and skips nucleotides as described above. However, the coalescent and shared memory techniques don’t apply to this version since they are specific to a GPU.

2.3 Programs compared and their versions and parameters

The literature contains many short read alignment programs that have been benchmarked extensively [10, 11]. Instead of considering many different programs we select the widely used program BWA [25] that uses the Burrows-Wheeler transform. We also select NextGenMap that uses hash-tables and is shown to be accurate on reads up to 10% mismatches compared to other leading programs [12]. We use the multi-threaded version of BWA and enable the GPU option in NextGenMap.

Other GPU programs for mapping short reads [26, 27, 28, 29] are implementations of CPU counterparts designed for speedup and achieve the same accuracy. Since they offer no improvement in accuracy they would perform poorly on divergent reads. Furthermore, the CPU program runtimes are already in seconds vs minutes and hours for exact methods (such as ours and Smith-Waterman) and so we exclude these programs from the paper.

From the category of exact mapping programs we use SSW [16] that uses a fast Single-Instruction-Multiple-Data (SIMD) Smith-Waterman algorithm to align a given read to the entire genome and the fast GPU Smith-Waterman program CUDA-SW++ [17]. As noted earlier this is designed for protein sequence database search and not for aligning to large genome sequences. However, we adapt it to short read mapping by considering fragments of the genome as database sequences and read as the query.

Below we describe program parameters and how we optimized them where applicable. The exact command line of each program is given in the Online Supplementary Material at [http://www.cs.njit.edu/usman/MaxSSmap](http://www.cs.njit.edu/usman/MaxSSmap).

**MaxSSmap**  For MaxSSmap we consider fragment lengths of 48 for E.coli genome and 480 for human chromosome one, match cost of 5 and mismatch cost of -4. In the exact alignment phase where we perform Needleman-Wunsch we consider the same match and mismatch cost and a gap cost of -26. We selected fragment lengths to optimize runtime. For E.coli genome we considered sizes 16, 32, 48, 64, and 80, and for human chromosome one we looked at 160, 240, 320, 400, and 480. The match and mismatch costs are optimized for accuracy on the 251bp length E.coli reads. For other genomes we recommend the user to experiment with different fragment sizes starting with a small value. As explained earlier when describing MaxSSmap the fragment length must be a multiple of 16 because of byte packing to allow storage of the genome in an array of int4 instead of char.
MaxSSmap_fast  In this faster version of MaxSSmap we consider 4 nucleotides in the read sequence when mapping to the genome, skip the next 12, and continues this cycle (as described earlier). The above version skips every other nucleotide instead. We consider a read mapped if the ratio of the best scoring fragment to the second highest one is below 0.85 (as opposed to 0.9 in MaxSSmap). The other parameters are the same as MaxSSmap.

SSW  This is a recent Smith-Waterman library that uses Single-Instruction-Multiple-Data (SIMD) to achieve parallelism. It has been shown to be faster than other SIMD based Smith-Waterman approaches [16]. It has also been applied to real data as a secondary program to align reads rejected by mainstream programs [16].

CUDA-SW++  CUDA-SW++ [17] is originally designed for protein database search. It performs Smith-Waterman alignment of the query to each sequence in the database in parallel. We simulate short read mapping with it by dividing the genome into same size disjoint fragments and considering each fragment of the genome as one of the database sequences and the read as the query. We set CUDA-SW++ to output the two top highest scoring fragments and their scores. If the ratio of the second best score to the best one is above .9 we do not consider the read mapped. We set the fragment length to 512 for the E.coli genome and the gap open and extension costs to -26 and -1, and the match and mismatch costs to 5 and -4. These values yielded highest accuracy for the simulated reads. We modified the code so that the blosum45 matrix uses +5 for match and -4 for mismatch. We choose this fragment length because lower ones reduce the runtime marginally but the accuracy goes down considerably whereas higher fragment lengths don’t yield higher accuracy and increase runtime. The gap, match, and mismatch costs are optimized for accuracy on the 251bp E.coli reads.

BWA-MEM  We use BWA-MEM version 0.7.5a with multi-threaded enabled (-t 12) and other options set to their default values.

NextGeneMap  We use NextGeneMap version 0.4.10 with the options -g 0 that enables the GPU and everything else default.

Meta-methods  We consider four meta-methods that first apply a fast mapper and then a slower yet more accurate aligner for rejected reads.

- NextGenMap + MaxSSmap
- NextGenMap + MaxSSmap_fast
- NextGenMap + CUDASW++
- NextGenMap + SSW

We use the same options for each program in the meta-method as described above.
2.4 Experimental platform

All programs were executed on Intel Xeon X5650 machines with 12GB RAM each equipped with three NVIDIA Tesla M2050 GPUs with 3GB global memory and 49152 bytes of shared memory. We used CUDA release 4.2 to develop MaxSSmap and to compile and build the GPU programs. In Table 1 we list the architecture on which we run each program.

| Program       | Architecture                             |
|---------------|------------------------------------------|
| MaxSSmap      | GPU                                      |
| MaxSSmap_fast | GPU                                      |
| CUDA-SW++     | GPU                                      |
| SSW           | SIMD single CPU (GPU unavailable)       |
| NextGenMap    | GPU                                      |
| BWA-MEM       | multi-threaded 12 CPUs                   |

Table 1: Architecture for each program compared in our study

2.5 Data simulation

We use the program Stampy [1] (version 1.0.22) to simulate reads with realistic base qualities. We use the E.coli genome K12 MG1665 (4.6 million bp) from which Stampy simulates reads and Illumina MiSeq 251bp reads in SRR522163 from the NCBI Sequence Read Archive (http://www.ncbi.nlm.nih.gov/Traces/sra) from which Stampy simulates base qualities. Roughly speaking each divergence corresponds to fraction of mismatches in the reads after accounting for sequencing error. For example .1 divergence means on average 10% mismatches excluding sequencing errors. We simulate one million 251 bp E.coli reads of divergences 0.1, 0.2, and 0.3 with and without gaps ranging up to length 30. The gaps are randomly chosen to occur in the read or the genome.

We also simulate 250bp reads divergences 0.1, 0.2, and 0.3 with and without gaps from human chromosome one (249 million bp). The chromosome sequence was obtained from the Genome Reference Consortium (http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc) version GRCh37.p13. We use Illumina MiSeq 250bp reads in ERR315985 through ERR315997 from the the NCBI Sequence Read Archive from which Stampy simulated base qualities. See Table 2 for exact Stampy command line parameters for simulating the data.

2.6 Measure of accuracy and error

For Stampy simulated reads the true alignment is given in a CIGAR string format [22]. Except for CUDA-SW++ we evaluate the accuracy of all programs with the same method used in [1]. We consider the read to be aligned correctly if at least one of the nucleotides in the read is aligned to the same one in the genome as given by the true alignment. It’s not unusual to allow a small window of error as done in previous studies (see [1] for a thorough discussion).
CUDA-SW++ does not output in SAM format. Instead it gives the top scoring fragments and the score of the query against the fragment. To evaluate it’s accuracy we divide the true position by the fragment size which is 512 in all our experiments. We then consider the read to be mapped correctly if the difference between the CUDA-SW++ fragment and the true one is at most 1.

3 Results

We study the accuracy and runtime of all programs and the four meta-methods described earlier. We measure their performance for mapping simulated E.coli and human reads to the E.coli and human chromosome one respectively.

3.1 Comparison of MaxSSmap and Smith-Waterman for mapping divergent reads to E.coli genome

We begin by comparing MaxSSmap and MaxSSmap_fast to SSW and CUDA-SW++. We map 100,000 251bp simulated E.coli reads to the E.coli genome. We simulate these reads using the
As mentioned earlier, MaxSSmap offers no advantage over mainstream approaches on reads with fewer than 10% mismatches. Programs like NextGenMap designed for mapping to polymorphic genomes can align such reads very quickly with high accuracy. Thus we consider three levels of divergence in the reads: 0.1, 0.2, and 0.3. Roughly speaking each divergence corresponds to the percentage of mismatches in the data.

| Div | MSS  | MSS_fast | CUDA-SW++ | SSW  |
|-----|------|----------|-----------|------|
| .1  | 95 (0.4) | 90 (0.2) | 94 (0.9) | 97 (3) |
| .2  | 94 (0.6) | 81 (0.6) | 94 (1)  | 97 (3) |
| .3  | 90 (1.1) | 51 (1.4) | 93 (1.3) | 96 (4) |

(a) Percent of 100,000 251bp reads mapped correctly to the E.coli genome. Shown in parenthesis are incorrectly mapped reads and remaining are rejected.

| Div | MSS  | MSS_fast | CUDA-SW++ | SSW  |
|-----|------|----------|-----------|------|
| .1  | 20  | 12       | 164       | 1288|
| .2  | 20  | 12       | 164       | 1275|
| .3  | 20  | 12       | 164       | 1255|

(b) Time in minutes to map 100,000 251bp reads to the E.coli genome.

Table 3: Comparison of MaxSSmap and MaxSSmap_fast to a GPU and a SIMD high performance Smith-Waterman implementation. These reads are simulated by Stampy and also contain realistic base qualities generated from real data. Each divergence represents the average percent of mismatches in the reads. So 0.1 means 10% mismatches on the average. The gaps are randomly chosen to occur in the read or the genome and are of length at most 30. We abbreviate MaxSSmap by MSS.

In Table 3(a) we see that the MaxSSmap accuracy is comparable to SSW and CUDA-SW++ except at divergence 0.3 with gaps (our hardest setting). Table 3(b) shows that the MaxSSmap and MaxSSmap_fast runtimes are at least 60 and 100 times lower than SSW and 8 and 13 times lower than CUDA-SW++. This is where the real advantage of MaxSSmap lies: high accuracy and low error comparable to Smith-Waterman on reads up to 30% mismatches and gaps yet at a lower cost of runtime.

At high divergence and with gaps we expect Smith-Waterman to fare better in accuracy and error than our maximum scoring subsequence heuristic. For example at divergence 0.3 with gaps SSW is 14% and 67% better than MaxSSmap and MaxSSmap_fast in accuracy. Even though MaxSSmap_fast has much lower accuracy at high divergence we see later it is still fast and useful in conjunction with a fast mapper.

Recall that MaxSSmap detects and rejects repeats which are likely to be errors. We use the same technique in the CUDA-SW++ output. However, SSW does not appear to have such a strategy and so we see a higher error for it.
3.2 Comparison of meta-methods for mapping divergent E.coli reads

We now compare the accuracy and runtime of four meta-methods that use NextGenMap in the first phase of mapping and MaxSSmap, MaxSSmap_fast, CUDASW++, and SSW to align rejected reads in the second phase. We study the mapping of one million 251 bp reads simulated E.coli reads to the E.coli genome.

The accuracy of NGM+CUDASW++ and NGM+SSW are comparable to NGM+MaxSSmap but runtimes are much higher. For example at divergence 0.2 with gaps NGM+SSW takes over 24 hours to finish and NGM+CUDASW++ takes 756 minutes, whereas NGM+MaxSSmap and NGM+MaxSSmap_fast finish in 68 and 47 minutes respectively. At divergence 0.3 with gaps both NGM+MaxSSmap and NGM+MaxSSmap_fast finish within two hours whereas both NGM+CUDASW++ and NGM+SSW take more than 24 hours. We choose the two fastest meta-methods for comparison to BW A and NextGenMap.

| Div | NGM+ MSS | NGM+ MSS_fast | NGM+ CSW++ | NGM+ SSW |
|-----|----------|---------------|------------|----------|
| .1  | 96 (1.3) | 92 (1.2)      | 97 (1)     | 97 (2.8) |
| .2  | 95 (1.5) | 87 (1.3)      | 96 (1)     | NA       |
| .3  | 91 (1.4) | 60 (1.6)      | 93 (1.5)   | NA       |

(a) Percent of one million 251bp reads mapped correctly to the E.coli genome. Shown in parenthesis are incorrectly mapped reads and remaining are rejected.

| Div | NGM+ MSS | NGM+ MSS_fast | NGM+ CSW++ | NGM+ SSW |
|-----|----------|---------------|------------|----------|
| .1  | 25       | 20            | 197        | 1397     |
| .2  | 57       | 42            | 537        | NA       |
| .3  | 148      | 110           | 1343       | NA       |

(b) Time in minutes to map one million 251bp reads to the E.coli genome.

Table 4: Comparison of meta-methods. See Table 3 caption for details about reads. We abbreviate NextGenMap by NGM, MaxSSmap by MSS, and CUDA-SW++ by CSW++.

3.3 Comparison of fastest meta-methods to NextGenMap and BWA for mapping divergent E.coli reads

In Table 5 we compare the accuracy and runtimes of NextGenMap and BWA to NGM+MaxSSmap and NGM+MaxSSmap_fast. NGM+MaxSSmap achieves high accuracy and low error at all settings but at the cost of increased runtime compared to NextGenMap and BWA. NGM+MaxSSmap_fast sacrifices accuracy for speed. On reads of divergence 0.1 and 0.2 with gaps it yields an improvement of 9% and 14% over NextGenMap while adding 24 and 45 minutes to the NextGenMap time of 1.5 and 2 minutes respectively. The runtimes for both meta-methods increases with higher divergence because there are many more reads rejected by NextGenMap at those divergences.

NextGenMap has higher accuracy than BWA as shown here (Table 5) and in previous studies [12] while BWA is the fastest program amongst all compared. We ran BWA in a multi-threaded mode.
that utilizes all CPU cores and all other methods on the GPU. We found that running NextGenMap on the GPU was faster than its multi-threaded mode.

### 3.4 Comparison of meta-methods to BWA and NextGenMap for mapping divergent reads to human genome chromosome one

We demonstrate runtimes for mapping one million 250bp reads created with the Stampy program (parameters given earlier) to the human chromosome one. We compare accuracy, error, and runtimes of BWA and NextGenMap to NextGenMap+MaxSSmap and NextGenMap+MaxSSmap_fast for mapping these reads.

In Table 5(b) we see that the runtime of the meta-methods are much higher than NextGenMap and BWA at divergence 0.2 and 0.3. At these settings there are many more rejected reads than at divergence 0.1. Mapping to the human genome chromosome one is more expensive for MaxSSmap because there are many more fragments to consider compared to E.coli genome. This puts MaxSSmap at a disadvantage but compared to Smith-Waterman alignment it is a much faster alternative.

At divergence 0.3 both MaxSSmap meta-methods take over 24 hours to finish. Table 6 shows that at divergence 0.2 with gaps (a hard setting) NGM+MaxSSmap and NGM+MaxSSmap_fast add 13% and 5% accuracy (with low error) and take a total of 1126 and 659 minutes compared to NextGenMap runtime of 51 minutes.

### 3.5 Comparison to parallel multi-threaded CPU implementation of MaxSSmap

We also study the runtimes of the parallel multi-threaded CPU implementation of MaxSSmap as described earlier. We examined three fragments lengths of 4800, 48000, and 480000. Each yields

| Div | NGM+ MSS | NGM+ MSS_fast | NGM | BWA |
|-----|----------|----------------|-----|-----|
| .1  | 96 (1.3) | 92 (1.2)       | 87  | 89  |
| .2  | 95 (1.5) | 87 (1.3)       | 72  | 24  |
| .3  | 91 (1.4) | 60 (1.6)       | 26  | 6.0 |

(a) Percent of one million 251bp reads mapped correctly to the E.coli genome. Shown in parenthesis are incorrectly mapped reads and remaining are rejected.

| Div | NGM+ MSS | NGM+ MSS_fast | NGM | BWA |
|-----|----------|----------------|-----|-----|
| .1  | 25       | 20             | 1.2 | 0.7 |
| .2  | 57       | 42             | 1.9 | 0.5 |
| .3  | 148      | 110            | 2.1 | 0.4 |

(b) Time in minutes to map one million 251bp reads to the E.coli genome.

### Table 5: Comparison of meta-methods to NextGenMap and BWA. See Table 3 caption for details about reads. We abbreviate NextGenMap by NGM and MaxSSmap by MSS.
### Table 6: Comparison of meta-methods to NextGenMap and BWA.

See Table 3 caption for details about reads. NA denotes time greater than 24 hours which is beyond our current computational resources. We abbreviate NextGenMap by NGM and MaxSSmap by MSS.

| Div | NGM+ | NGM+ | NGM | BWA |
|-----|------|------|-----|-----|
|     | MSS  | MSS  |     |     |
| Reads without gaps | | | | |
| .1 | 99 (1) | 99 (1) | 99 (1) | 96 (3) |
| .2 | 94 (6.2) | 93 (6.2) | 86 (6) | 33 (6) |
| .3 | NA | NA | 37 (9) | 0.8 (0.6) |
| Reads with gaps | | | | |
| .1 | 98 (2) | 97 (2) | 96 (2) | 89 (3) |
| .2 | 92 (6.5) | 84 (6.3) | 79 (6) | 28 (5) |
| .3 | NA | NA | 30 (7) | 0.7 (0.5) |
| Reads with gaps | | | | |
| .1 | 176 | 104 | 9.8 | 1.5 |
| .2 | 1126 | 659 | 51 | 0.7 |
| .3 | NA | NA | 66 | 0.6 |

(a) Percent of one million 250bp reads mapped correctly to the human chromosome one genome. Shown in parenthesis are incorrectly mapped reads and remaining are rejected.

(b) Time in minutes to map one million 250bp reads to the human chromosome one genome.

4 Discussion

In our experimental results we have demonstrated the advantage of MaxSSmap over Smith-Waterman for mapping reads to genomes. In scenarios where accurate re-alignment of rejected and low-scoring reads are required MaxSSmap and MaxSSmap.fast would be fast alternatives to Smith-Waterman. Such conditions are likely to contain reads with many mismatches and gaps which would get rejected by mainstream programs.

Our program, however, is not without limitations. For mapping to human chromosome one the runtimes are higher than for E.coli just because there are many more genome fragments for the former. We also see that the accuracy of MaxSSmap is lower than Smith-Waterman as we cross into higher divergence of 0.3 with gaps.

5 Conclusion

We introduce a GPU program called MaxSSmap for mapping reads to genomes. We use the maximum scoring subsequence to identify candidate genome fragments for final alignment instead
of hash-tables and Burrows-Wheeler transform. We show that MaxSSmap has comparable high accuracy to Smith-Waterman based programs yet has lower runtimes and accurately maps reads rejected by a fast mainstream mapper faster than if Smith-Waterman were used.

6 Acknowledgements

We thank Peicheng Du and Jeffrey Rosenfeld for helpful suggestions to our paper. This research was performed on a GPU cluster at UNC Charlotte thanks to Dennis R. Livesay and a GPU machine at NJIT thanks to Shahriar Afkhami. Funding was provided by the National Science Foundation Assembling Tree of Life grant 0733029.

References

[1] Gerton Lunter and Martin Goodson. Stampy: A statistical algorithm for sensitive and fast mapping of illumina sequence reads. Genome Research, 21(6):936–939, 2011.

[2] Zhong Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. Nat Rev Genet, 10(1):57–63, 2009.

[3] Vinicio Reynoso and Catherine Putonti. Mapping short sequencing reads to distant relatives. In Proceedings of the 2nd ACM Conference on Bioinformatics, Computational Biology and Biomedicine, BCB ’11, pages 420–424, 2011.

[4] Lesley J. Collins, Patrick J. Biggs, Claudia Voelkel, and Simon Joly. An approach to transcriptome analysis of non-model organisms using short-read sequences. Genome Informatics, 21:3–14, 2008.

[5] Christopher M. Seabury, Eric K. Bhattacharai, Jeremy F. Taylor, Ganesh G. Viswanathan, Susan M. Cooper, Donald S. Davis, Scot E. Dowd, Mitch L. Lockwood, and Paul M. Seabury. Genome-wide polymorphism and comparative analyses in the white-tailed deer (italic¡odocoileus virginianus¡/italic): A model for conservation genomics. PLoS ONE, 6(1):e15811, 01 2011.

[6] Chunguang Liang, Alexander Schmid, Maria Lopez-Sanchez, Andres Moya, Roy Gross, Jorg Bernhardt, and Thomas Dandekar. Jane: efficient mapping of prokaryotic ests and variable length sequence reads on related template genomes. BMC Bioinformatics, 10(1):391, 2009.

[7] Kin Fai Au, Jason G. Underwood, Lawrence Lee, and Wing Hung Wong. Improving pacbio long read accuracy by short read alignment. PLoS ONE, 7(10):e46679, 10 2012.

[8] Tami D Lieberman, Kelly B Flett, Idan Yelin, Thomas R Martin, Alexander J McAdam, Gregory P Priebe, and Roy Kishony. Genetic variation of a bacterial pathogen within individuals with cystic fibrosis provides a record of selective pressures. Nat Genet, 46:82–87, 2014.
[9] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nat Genet*, 44:226 – 232, 2012.

[10] Nuno A. Fonseca, Johan Rung, Alvis Brazma, and John C. Marioni. Tools for mapping high-throughput sequencing data. *Bioinformatics*, 28(24):3169–3177, 2012.

[11] Ayat Hatem, Doruk Bozdag, Amanda Toland, and Umit Catalyurek. Benchmarking short sequence mapping tools. *BMC Bioinformatics*, 14(1):184, 2013.

[12] Fritz J Sedlazeck, Philipp Rescheneder, and Arndt von Haeseler. Nextgenmap: Fast and accurate read mapping in highly polymorphic genomes. *Bioinformatics*, 2013.

[13] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[14] J. Bentley. *Programming Pearls*. Addison-Wesley, 1986.

[15] Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Trans. Program. Lang. Syst.*, 7:113–136, January 1985.

[16] Mengyao Zhao, Wan-Ping Lee, Erik P. Garrison, and Gabor T. Marth. Ssw library: An simd smith-waterman c/c++ library for use in genomic applications. *PLoS ONE*, 8(12):e82138, 12 2013.

[17] Yongchao Liu, Douglas Maskell, and Bertil Schmidt. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. *BMC Research Notes*, 2(1):73, 2009.

[18] NVIDIA. CUDA C Programming Guide 4.0. [http://developer.nvidia.com/cuda-toolkit](http://developer.nvidia.com/cuda-toolkit), 2011.

[19] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition, 2010.

[20] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[21] Brent Ewing, LaDeana Hillier, Michael C. Wendl, and Phil Green. Base-calling of automated sequencer traces using phred. i. accuracy assessment. *Genome Research*, 8(3):175–185, 1998.

[22] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Yue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[23] NVIDIA. CUDA C Best Practices 4.0. [http://developer.nvidia.com/cuda-toolkit](http://developer.nvidia.com/cuda-toolkit), 2011.
[24] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, 2008.

[25] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrowswheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[26] Petr Klus, Simon Lam, Dag Lyberg, Ming Cheung, Graham Pullan, Ian McFarlane, Giles Yeo, and Brian Lam. Barracuda - a fast short read sequence aligner using graphics processing units. *BMC Research Notes*, 5(1):27, 2012.

[27] Michael Schatz, Cole Trapnell, Arthur Delcher, and Amitabh Varshney. High-throughput sequence alignment using graphics processing units. *BMC Bioinformatics*, 8(1):474, 2007.

[28] Yongchao Liu, Bertil Schmidt, and Douglas L. Maskell. Cushaw: a cuda compatible short read aligner to large genomes based on the burrowswheeler transform. *Bioinformatics*, 28(14):1830–1837, 2012.

[29] Chi-Man Liu, Thomas Wong, Edward Wu, Ruibang Luo, Siu-Ming Yiu, Yingrui Li, Bingqiang Wang, Chang Yu, Xiaowen Chu, Kaiyong Zhao, Ruiqiang Li, and Tak-Wah Lam. Soap3: ultra-fast gpu-based parallel alignment tool for short reads. *Bioinformatics*, 28(6):878–879, 2012.