Optimizing Bioinformatics Operations using High Performance Computing

Prepared By

Parisha Jindal

12MCEC09

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2014

Optimizing Bioinformatics Operations using High Performance Computing

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

Prepared By

Parisha Jindal

(12MCEC09)

Guided By

Prof Monika Shah

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2014

Undertaking for Originality of the Work

I, **Parisha Jindal**, Roll. No. **12MCEC09**, give undertaking that the Major Project entitled "**Optimizing Bioinformatics Operations using High Performance Computing**"submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering** of Nirma University, Ahmedabad, is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

Endorsed by

Prof. Monika Shah

(Signature of Guide)

Certificate

This is to certify that the Major Project Report entitled "**Optimizing Bioinformatics Operations using High Performance Computing"** submitted by **Parisha Jindal (Roll No: 12MCEC09)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project part, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Guide: Prof Monika Shah
Assistant Professor,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Prof. Vijay Ukani
Associate Professor
Coordinator M.Tech - CSE
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Sanjay Garg
Professor and Head,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr K Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Prof Monika Shah**,Professor, Computer Science Department, Institute of Technology, Nirma University, Ahmedabad for her valuable guidance and continual encouragement throughout this work. The appreciation and continual support she has imparted has been a great motivation to me in reaching a higher goal. Her guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

My deepest gratitude is extended to **Prof. Vijay Ukani**, PG CSE - Coordinator, Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad for support and continual encouragement throughout the Major Project.

It gives me an immense pleasure to thank **Dr. Sanjay Garg**, Head of Computer Science and Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr K Kotecha**, Director, Institute of Technology, Nirma University, Ahmedabad for the motivation he has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, and Ahmedabad for their special attention and suggestions towards the project work.

The blessings of God, my family members and friends make the way for completion of Project. I am very much grateful to them.

- Parisha Jindal

12MCEC09

Abstract

The continuous increase in population has exponentially increased the biological datasets to be processed. Hence computations to process these datasets have also increased tremendously. An expedited solution for analysis of this data is required to enable quick decision making for various researches and medical treatments. This raise demand for HPC based solutions of bioinformatics operations for quick processing. Major focus of optimization is on the operations of DNA assembly and alignment. Many companies like Xcelris Genomics working in the field of bioinformatics and life sciences have identified the need for optimized solution for DNA assembly and alignment as time and space complexity of existing approaches delay research and analysis based on these datasets. This dissertation work focus on optimization of well known DNA sequence assembly tool Velvet using hybrid computing of HPC technologies.

Contents

List of Tables

List of Figures

Abbreviations

GPU ..... Graphics Processing Unit
HPC ..... High Performance Computing
DNA ..... Deoxyribonucleic acid
BP ..... Basepair (A,T,C,G nucleotides pairs)
DDNTP ..... Dideoxy nucleoside triphosphate (nucleotides lacking 3 prime end)
NGS ..... Next Generation Sequencing
WGS ..... Whole-genome shotgun
HGS ..... Hierarchical Shotgun Sequencing
BLAST ..... Basic Local Alignment Search Tool
MIMD ..... Multiple instruction, multiple data
SIMD ..... Single instruction, multiple data
MPI ..... Message Passing Interface
CUDA ..... Compute Unified Device Architecture
SM ..... Streaming Multiprocessors
CPU ..... Central processing unit
RAM ..... Random access memory

Chapter 1

Introduction

1.1 General

Computational methods for processing biological data has seen a tremendous development in the last few decades. A quick analysis of the biological data is required to provide solutions for researches and treatments in less time. For this we need time efficient bioinformatics operations. Bioinformatics is the use of computational methods aiming at processing and analysing biological data to provide solutions for decoding the information embedded in it. Major focus of bioinformatics is on the enhancement of operations of DNA assembly and alignment. Persistent research is going on to develop advance techniques which can provide time efficient solutions for these operations. The optimization approaches adopted for this purpose include:

- Algorithmic approach: Developing algorithms based on data ordering, data structures and techniques that can provide expedited solutions.
- HPC approach: Using parallel techniques like multicore and manycore for concurrently processing data.

Much work has been done to parallelize the existing algorithms. Many assembly and alignment techniques are now being implemented on parallel systems. Thus this research is motivated by need in the bioinformatics community for sequence assemblers and aligners and increasing use of HPC for research.

1.2 Objective

The objective of this research is to optimize the bioinformatics operation of DNA assembly which is merging of short fragments of DNA to reconstruct the original complete sequence by parallelization on GPU. This project aims at optimizing the velvet assembler code by implementation on GPU.

1.3 Motivation

This work is motivated by a number of factors, most important of which is providing a real time, accelerated solution of DNA assembly. Another motivating factor is the increasing trend of multi-core computing in bioinformatics. It is expected to have solutions that can fully exploit the potential of the emerging paradigm of parallel computing in bioinformatics and revolutionize this research area.

1.4 Scope

This research will provide solutions for identifying, decoding and analyzing genes for species discovery and also will help in developing healthcare solutions by disease diagnosis (like TB, cancer etc by finding mutations/deviations in dna sequence from an infected dna).

1.5 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2 *Background of the Project*, describes various operations and applications of Bioinformatics. Also gives some background information for High Performance Computing and Graphic Processing Unit.

Chapter 3 *Literature Survey*, compares various techniques of DNA Assembly and significance of HPC in assembly.

Chapter 4 *System architecture of Velvet Assembler*, gives the workflow for velvet assembler.

Chapter 5 *Research Gaps and Optimization Scope in Velvet*, finds hotspots based on profiling done on velvet and discovers research gaps and optimization scope.

Chapter 6 *Proposed Work*, proposes solutions for research gaps optimization scopes identified .

Chapter 7 *Experiment Setup and Results Analysis*, gives the implementation results, speedups of parallel execution.

Finally, **Chapter 8** *Conclusion and Future Scope*, gives concluding remarks and scope for future work.

Chapter 2

Background for Project

2.1 Bioinformatics

Bioinformatics is an interdisciplinary field of study which aims at retrieving, improving and analyzing the biological data like DNA. Its major focus is on the development of software solutions for producing useful biological information. Bioinformatics uses areas of computer science, mathematics and engineering to process and analyze biological data. Bioinformatics is highly popular in decoding DNA sequences. Various bioinformatics operations that are commonly performed on DNA are sequencing, assembly, and alignment. Bioinformatics finds various applications in developing cures for diseases, identifying species etc.

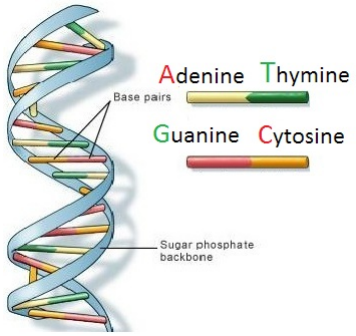

Figure 2.1: DNA structure

DNA is the basic structural block of life forms which is responsible for introducing diversity in organisms. DNA strand consist of four bases adenine, guanine, cytosine, and thymine which gives specific characteristic to each individual. Determining the order of these bases is performed through DNA sequencing. Several sequencing technologies

exist but they are limited and can process only short sequences. Sequencing the DNA involves randomly fragmenting long strands of DNA into small pieces and sequencing these fragments. Because of random breaking, most of them overlap and give information needed to combine them back together. The process of recreating the original DNA sequence from fragment reads is DNA assembly. Assembly is a computation intensive process and can take days depending on the size of DNA being processed.

2.2 Bioinformatics Operations

The three operations of Bioinformatics are interrelated. All these operations are important steps in computational biology and are a core component in decoding biological information.

2.2.1 DNA Sequencing

DNA sequencing is the process of finding the exact order of nucleotides/bases within DNA. The development of sequencing methods has accelerated research of medicine and biology. In 1975 Frederick Sanger developed the basic sequencing method that is still widely used today [2]. Currently, several different sequencing techniques exist: first generation techniques based on Sanger method, and next generation sequencing techniques (NGS). Next generation techniques provides high throughput at low cost.

Fundamental steps of Sequencing involve making many copies of the base DNA sequence of a single strand of DNA. Steps of Sequencing are as follows:[2]

- DNA is extracted from sample and divided into single strands.
- Then broken down into small fragments known as reads. (Shotgun Sequencing)
- The DNA fragments are mixed with four nucleotides to construct matching strands.
- ddntps: nucleotides lacking a 3' hydroxyl which are fluorescent are added.
- Many matching strands are thus constructed and construction stops when a ddntp is added to sequence.
- The mixture is then separated using electrophoresis which produces sequence of colored bands.

- The sequence of these bands determines sequence of the strand which can be converted to sequence of nucleotides in the original DNA using the matching rule (adenine pairs with thymine and cytosine with guanine).
- DNA sequencing results may be viewed on a computer screen as a set of colored peaks.
- The sequenced reads are then send to assembler which merges those reads using their overlapping information to get the sequence of complete DNA.

Applications of DNA Sequencing

- Forensics: To identify criminals, paternity of a child, endagered and protected species.
- Medicine: To detect genes associated with heredity or acquired diseases.
- Agriculture: To prepare hybrid varieties of crops and improve dairy production.

2.2.2 DNA Assembly

Sequencing technologies do not produce complete DNA, instead generate small subsequences of the whole DNA known as reads which can occur multiple times. The average number of times a nucleotide occurs across reads is called the coverage. Combining the fragments back into a single sequence of nucleotides by using information common between them is called DNA assembly, while the program that perform this process are called DNA assembler.

DNA assembly can be divided into two groups:

- De novo - Assembly of sequence reads into longer adjacent sequences, contigs, followed by the ordering contigs when reference DNA is not present.
- Mapping/reference - When available, reference DNA can be used to align the reads of a newly obtained DNA, which helps in determining accuracy of assembly.

This review work will be focusing mainly on de-novo assembly as Comparative approaches can be applied to the DNA for which reference sequence is available. After the de novo assembly, connected reads (contigs) are generated which are extended to get super-contigs or scaffolds using mate pair information and are placed in order to get

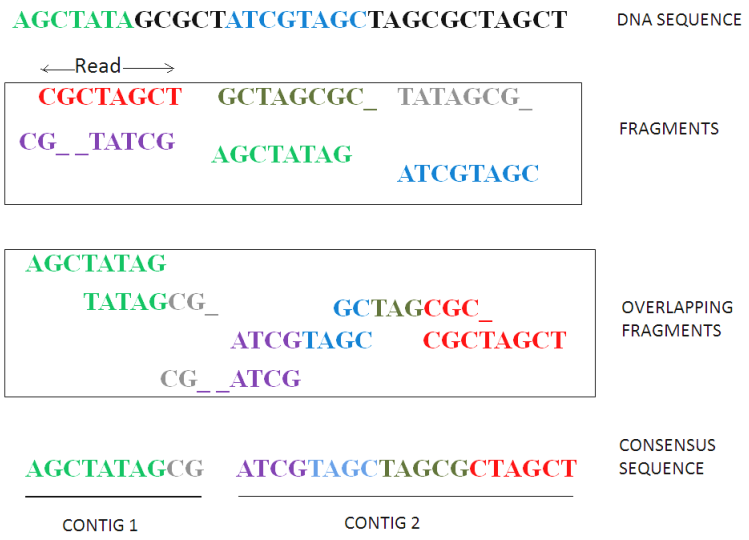

Figure 2.2: Assembling the reads

the assembled genome [3]. Sequencing of a DNA can be done by two ways: single end sequencing where only one strand is sequenced and the other is paired end sequencing where both strands are sequenced from their 5 prime ends [4]. Mate pair sequencing provides more information for recreation as it gives the distance between two reads.

Applications of DNA Assembly

- Aims to advance medicine and global health.
- Contribute directly to a scientific research to discover new species.
- Find similarities between two species.

2.2.3 DNA Alignment

It is the arrangement of DNA or protein sequences, to identify similar regions. It is used to infer relationship between the sequences. Even in de novo assembly, we may need sequence to be aligned back to the assembly to check the assembly accuracy. Methods of Sequence Alignment: There are mainly two methods of Sequence Alignment:

Alignment is commonly divided into two groups:[5]

- **Global Alignment**: Sequences which are of same length and similar are suitable for global alignment. Here, the alignment is carried out from beginning till end of the sequence to find out the best possible alignment.

- **Local Alignment**: Sequences having a possibility of similarity or even dissimilar sequences can be compared with local alignment. It finds regions with high similarity.

Techniques of Alignment

- **Dot Matrix:** Match in base is represented as dot. Very much similar sequences are represented as prominent diagonal. Good for detecting repeats but makes no distinction between mismatch and gaps.[6]
- **Dynamic programming:** Generates optimal/best alignment by giving scores to match/mismatch and gaps. Similar to longest common subsequence problem which finds area of similarities. Time and memory consuming but give accurate results.[6]
- **Word or k-tuple method:**In this case, the algorithm compares small sections, with some minimum word length, which are used to begin the alignment instead of comparing single letters. Eg. BLAST and FASTA.[7][6]

Applications of DNA Alignment

- Finds functional similarities between two species.
- Resolve paternity issues.
- Crime Scene Investigation.

2.2.4 Bioinformatics Operations Workflow

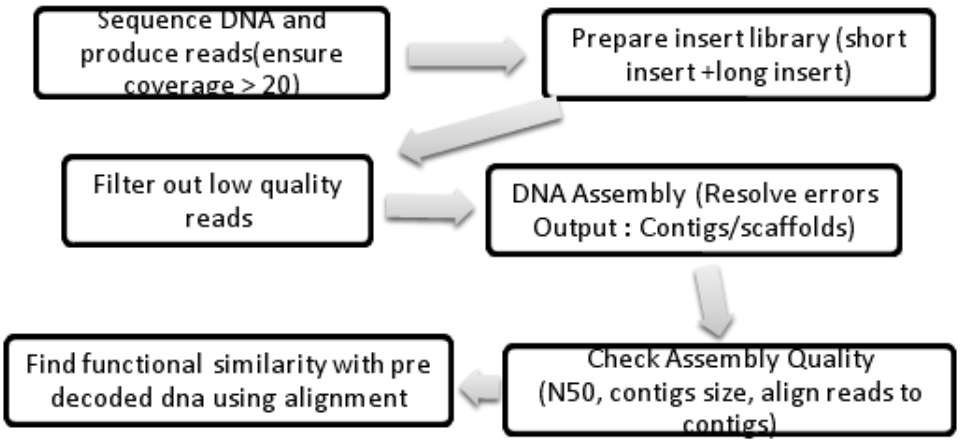

Figure 2.3: Bioinformatics Operations Workflow

2.3 High Performance Computing

High Performance Computing refers to exploiting computational power such that it delivers high performance than one could get from single computer to solve large problems in science and other research. Commonly available options of HPC are clusters, grids, GPU, clouds. There is lot of research going on in the field of many-core computing which utilizes the computation power of GPU. GPU computing is using GPU together with CPU to speedup general-purpose applications.

2.3.1 GPU Architecture

GPU is a compute device which is a co-processor to the CPU, has its own RAM, and runs multiple threads in parallel. Parallel portions of application are executed on GPU as kernels which run in parallel on many threads. Difference between GPU and CPU threads are: [1]

- GPU threads are extremely lightweight and require little creation overhead.
- GPU needs 1000s of threads for full efficiency where as multicore cpu needs only a few.

Kernel is executed in a grid of thread blocks which is a group of threads that can execute with each other by sharing data. There is a limit to the number of threads per block, since

all threads of a block reside on the same processor core and share the limited memory resources. The number of blocks in a grid is decided by the size of data being processed or number of processors in the system. Thread block can be executed in any order, in parallel or in series. This requirement allows thread blocks to be run in any order across any number of cores, enabling to write codes that scales with the number of cores. [1]

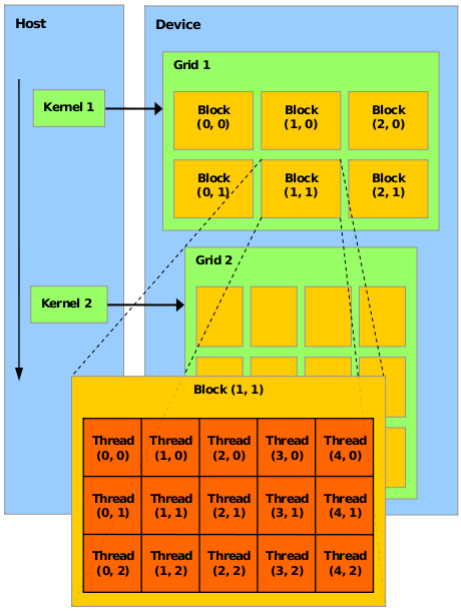

Figure 2.4: GPU Architecture[1]

CUDA Memory Model Each CUDA thread has private local memory. Each thread block has shared memory which is visible to all threads of the block. There are also two additional read-only memory spaces accessible by all threads: the constant and texture memory. All threads have access to the global memory. The global, constant, and texture memory spaces are optimized for different memory usages.

Hardware Implementation[1]

- Each thread block of grid is divided into warps which is the number of threads that can execute parallel and get executed by one multiprocessor.
- Each thread block is executed by one multiprocessor.

A multiprocessor can execute several blocks concurrently. Shared memory and registers are allocated among the threads of all concurrent blocks. So, decreasing shared memory usage and register usage increases number of blocks that can run concurrently.

Table 2.1: GPU Memory Components

Memory	Location	Accessible by	Lifetime
Registers	On chip	Thread	Thread
Local	Off chip	Thread	Thread
Shared	On chip	Block	Block
Global	Off chip	threads + host	Host Allocation
Constant	Off chip	threads + host	Host Allocation
Texture	Off chip	threads + host	Host Allocation

Shared memory is an on chip memory and is very fast. It is limited in size whereas global memory is large but slow. GPU offers high performance by transferring computation intensive tasks of the application to the GPU, while the serial code runs on the CPU. Pros and cons of GPU are given below:

Table 2.2: Pros and Cons of GPU

Pros	**Cons**
Higher Computation power : more ALU, more core Processor	Not suited to complex processing on single or few streams of data
Highly Parallel: many transistors	Not direct interface to I/O
Higher Bandwidth	Small cache so less data reusability
Lightweight threads	

HPC in Bioinformatics can prove to be very useful in processing the large volume of data generated in relatively less time. When done sequentially, DNA assembly takes many days. Even on cpu clusters a human dna assembly will take around 4-5 days. This time can be reduced if the processing can be transferred on GPU which has multiple fast processing cores which can do the processing parallely and will reduce time significantly. GPU is a better choice as it has lightweight threads which reduces time taken in thread switching.

Also GPU can be combined with clusters to resolve the problem of thread divergence to make use of all resources optimally.

Table 2.3: GPU vs CPU

	Clusters	**Grid**	**GPU**
Application	Dedicated,Computate intensive	Compute intensive, Provides range of services	Compute intensive operations
Suitable Class	MIMD	MIMD	SIMD
Processing	heavyweight	heavyweight	Lightweight Threads
Security Requirement	Low	High	Low
Speed (latency, bandwidth)	Low, high	High, Low	Low, high
Programming	MPI	MPI	CUDA

Chapter 3

Literature Survey

3.1 DNA Sequencing and Assembly

Sequencing technologies have improved considerably since the first DNA was read [2]. They are used to determine the DNA sequence of a new species or an individual. Despite success in determining the human[9] , and several other genomes, most species have not been sequenced yet. An important stage in DNA sequencing is the assembly of shotgun reads, i.e. putting together fragments randomly extracted from the sample to form a set of contiguous sequences or contigs. Algorithms were developed for whole genome shotgun (WGS) fragment assembly and have proved their utility through numerous genome assemblies.

Traditional Approach vs Next Generation Sequencing

The traditional method of sequencing, shotgun sequencing[9], is sequencing random fragments of the sample, then assembling them together. This works well on short and un-repetitive sequences, but is more difficult in long, repeat-rich DNA. Solution to it are:

- **Hierarchical shotgun sequencing** (The International Human Genome Sequencing Consortium, 2001). In this the DNA is biochemically subdivided into random regions which are mapped on DNA. The prefix and suffix are then compared to determine overlaps and construct contigs. These are then mapped onto the DNA. These clones are then sequenced separately.

- **Whole genome shotgun (WGS) sequencing**[9] determines the complete DNA sequence of an organism's DNA at a single time without using any intermediate

maps. This approach makes the assembly much more complex, as all the reads have to be processed together. It requires thoroughly accurate algorithm to avoid misassembles.

Recently new sequencing methods have emerged commonly known as next generation sequencing. Those available are pyrosequencing (454 Sequencing) , sequencing by synthesis (Illumina) and sequencing by ligation (SOLiD) [10]. Compared to traditional Sanger sequencing, these ngs technologies have lower costs and high throughput but reads produced by these technologies are much shorter than traditional Sanger reads. Because of their length, they must be produced in large quantities and at greater coverage depths. Whereas long reads provide long overlaps, to disambiguate repeats from real overlaps, short reads within repeats offer fewer differences to judge from. These issues have led to design of de novo assembly tools specifically for these short reads.[11]

3.1.1 De-novo DNA Assembly

Traditional methods vs. De Bruijn graph based approaches

Many algorithms have been proposed for solving problem of DNA assembly. The first traditional approach to assembly was Greedy Method including greedy string-based methods which was similar to finding longest common subsequence. In this reads are first checked for overlaps and overlaps are given a score and those with highest scores are merged. This was discontinued as it did not took into account the repeat structures within reads and produced mis-assemblies.

This traditional approach was replaced by the overlap layout consensus graph. In this the reads are marked as nodes in graph and there is an edge between two nodes if they have overlapping regions. A Hamiltonian path is then found to give a complete DNA sequence. This was used until recently but had been replaced by de-bruijn graphs as it became computationally complex as the number of reads grew resulting in drastic increase in number of nodes in graph.

In 1995, [12] introduced the use of a sequence graph to represent an assembly which could detect all the k nucleotide words, known as k-mers, present in a given genome. A node is created for every k-mer, then connect the nodes corresponding if the k-mers are overlapping. They could then report chains of overlapping k-mers which unambiguously produced contigs, because of an absence of branching connections. Pevzner et al. [13]

expanded on this idea. This approach was first presented in 2001 in which reads are broken down first into subsequences of fixed size k-mers. Next, the assembler builds a directed deBruijn graph in which each edge corresponds to a k-mer from one of the original sequence reads. The source and destination nodes correspond respectively to the k − 1 prefix and k − 1 suffix of the corresponding k-mer. Then a Eulerian tour is performed to give a continuous sequence. The de Bruijn approach was made popular by the Euler [14] assembler and is the main core for the design of modern assemblers targeted at short-read sequencing data, such as Velvet [15],Abyss[16] and ALLPATHS[17]. The chart gives a comparison of the three approaches to assembly.

Table 3.1: De-novo Assembly Algorithms Comparison

Approach	**Description**	**Assembler**	**Pros**	**Cons**
Greedy[18]	Iterative Reads with largest overlaps merged	Cap3, Ssake Sharcgs, Vcake	Computationally easy	Produce several mis-assemblies due to repeat
Overlap-Layout-Consensus [18]	Graph nodes reads, edges overlap. Determine Hamiltonian path	Celera ,Arachne,	Suitable for low coverage, long reads	Not suitable for long DNA sequences
De-Bruijn Grpah [13]	Reads broken into k-mers Debruijn graph, Vertices as k-mers, edge between vertices if from same read and have overlapping subsequence of length (k-1) in between. Then finds eulerian tour.	Velvet , Allpaths, Abyss	Suitable for high coverage, short reads, Fast, Sensitive to sequencing errors	Not read coherent, memory intensive

Measures of assembly accuracy [19] Some of the most important parameters to measure assembly are:

- N50: Contig length such that 50 percent DNA lies in block of this size or larger, gives the measure of DNA covered by large contigs. Larger contigs mean efficient assembly.
- DNA coverage: Percent of bases in reference covered by assembled contigs
- Maximum Contig size: Largest contig created

Challenges in Assembly

All these results in error in the sequencing mechanism:

a. Repeats: multiple near-identical copies throughout the genome

b. Incomplete Coverage: Some of the bases might not be covered while sequencing which result in problems in assembly due to incomplete information.

c. Insertion/Deletion of bases: Some bases might get deleted while there might be insertion of new bases due to contamination

3.1.2 Significance of HPC in DNA Assembly

Hpc provides low cost, time effective solutions for sequence assembly by distributing work to be done parallel. One of the efficient implementation in assembly using GPU is kmer generation in parallel. The reads generated are divided across threads which create a distributed spectrum of kmers. These k-mers are then are stored in hash tables. From this de-bruijn graph is constructed, unambiguous k-mers are merged followed by an Euler tour to give consensus sequence. Error correction is generally implemented as an integrated step.

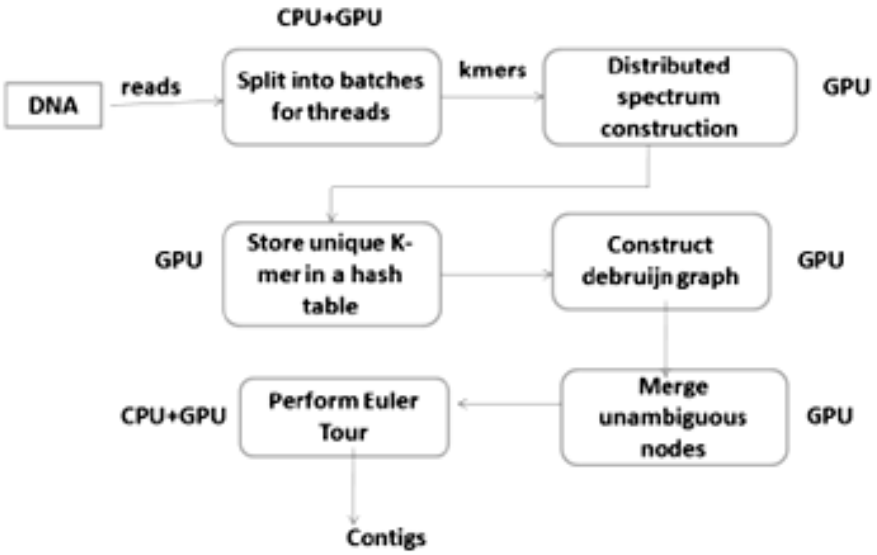

Figure 3.1: Denovo Assembly using HPC

Some assemblers using HPC solutions for DNA assembly are Velvet (openmp), Abyss (mpi) , SoapDenovo (openmp), Pasha. Among these velvet is the most liked assembler as it gives more accurate assemblies for both long and short reads. The following table gives a comparison for the parallel assemblers with de-bruijn graph approach. GPU

Table 3.2: Parallel Assemblers

Assembler	Parallel Approach	K-mer Hashing
PASHA [20]	multicore+MPI	Google Sparse Hash library
ABYSS [16]	MPI	Google Sparse Hash library
RAY [21]	MPI	Splay tree
PASQUAL [22]	multicore	Bloom Filter
VELVET [15]	multicore	Splay tree

computational methods has been applied in other procedures like in error correction and speeded up of these processes many times. Great optimization is expected in assembly using HPC tools

3.2 Kmer Hashing

To achieve high performance on GPUs, besides increasing parallelization we need data structures which enable fast insertions and retrievals. These should not be such that access to them deteriorates GPUs performance. Many data structures have been tested on GPU like trees, queues, linked list, hash tables, etc. Finding most efficient and parallelizable data structure suited to different algorithm and dataset is an important research challenge. For the process of DNA assembly, preliminary step involves counting k-mers to find unique and repeated k-mers. Counting of k-mers require a large data structure whose size will be proportional to the number of k-mers occurring in DNA read file which can result in memory overflows. Thus we need data structures that can fit into predefined disk space. For finding path from one k-mer to other for contig creation, a mapping of repeated k-mer to its first occurrence is required. Majority of assemblers use hash tables for counting k-mers as they provide quick random access to sparse data. Velvet uses Splaytable [15] as hash table for computing overlapping regions between two sequences. Within each node of Splaytable is a splaytree structure for handling collisions. This method is not suitable for implementation on GPUs as it has multiple entries at same node and thus access to one entry will require to go through all nodes leading to sequential flow. Therefore a hash implementation suitable for GPU is required.

A hash function should be such that it generates minimum collision. However if collision occurs, there should be provision for handling these colliding values and storing them in the hash table. Methods like open addressing and chaining are available for collision resolution however these are suited only for serial applications. In parallel environments

these leads to one thread waiting for another thread which results in serialization. Also the data access should be coalesced such that threads in a warp access nearby data so that there are minimum cache misses, but hash tables exhibit little locality of reference.

3.2.1 Work done for parallelization of Hashing

[23] have suggested a collision free hash mechanism known as perfect hashing which creates almost n^2 slots where n is the number of data elements to be hashed. Cuckoo Hashing by [24] makes use of multiple hash tables for insertion. If hash table location at one index found by hash function one is not empty it is hashed to another hash table. If all hash location at index are full than the element to be inserted evicts previous key from one of the sub table and inserts itself. The evicted element then becomes the current key and attempts to insert it are then done. The parallel version inserts data simultaneously iterating through sub tables simultaneously. Coherent parallel hashing by [25] exploits coherence in the data for faster performance. It exhibits much greater locality of memory accesses and consistent execution paths within groups of threads. GPU-based Locality Sensitive Hashing for K-Nearest Neighbor [26] uses the Bi-level LSH algorithm, which computes k-nearest neighbors. Bloom filters offers low memory consumption technique for hashing [27]. In this two hash functions are calculated and the positions given by them are set to 1. To query an element the bits at two hash positions are checked and an element is present if all bits are 1 else it is not present. However this method can result in false positive results. Most popular hashing algorithm is suggested by Alacantra et. al. in Real-Time Parallel Hashing [28] on the GPU which makes use of both perfect hashing and cuckoo hashing which resolves the problem of both collision and fast build up time. The following chart gives a summarized comparison of various parallel hashing mechanisms.

Hash table construction for counting k-mers is a tedious process and buildup time is proportional to number of unique k-mers. These reads can easily reach millions in number resulting in memory overflows on GPUs. Thus we need a hash table implementation which gives fast insertion while working in predefined memory space. There is no coherence between the k-mers i.e they are not dependent on their locations within read. Therefore Coherent hashing and Fast GPU based LSH are not suitable for hashing reads. Perfect hashing allocates n2 space which will result in memory leaks as GPU has very limited

memory. We cannot use Bloom filters because it only stores bits for presence but for storing k-mers we need information like readid, position. Most suitable approach here is to use Real time parallel hashing modified to suit our data as it is suitable for dynamic data. The table gives summary of Parallel Hashing Techniques

Paper	Technique	Advantage	Bottlenecks
Real-Time Parallel Hashing on the GPU [28]	• Involves both perfect hashing and cuckoo hashing. • Partition data in buckets of 512 elements and within each bucket parallel cuckoo hash places items into subtables.	Cuckoo hashing performed entirely on on-chip memory thus very fast. Maximum probes required is 3, one for each sub table.	Rebuilding of tables in case of insertion failure is time consuming.
Perfect Spatial Hashing [23]	• Create a hash table of approximately size n^2 so that all items can be accommodated.	Each element can be accessed in O(1) time	Size of data should be previously known. Too much space consuming.
Coherent Parallel Hashing [25]	• To insert key iterate until empty location is found. Number of steps for successful insertion is age of key. Inserts,difficult to insert keys by replacing keys with greater age by key with less age. • Preserves coherence by making sure neighboring values test neighboring locations.	Low failure rate at high load and provides fast queries	In absence of coherence among data this does not give good performance.Extra memory required for storing key ages.
Fast GPU-based LSH for KNN Computation [26]	• First partition data into groups so that similar are clustered together. • Then compute the Bi-Level LSH code for each item and construct hash table using parallel cuckoo hashing.	Uses radix sorting, which can benefit the high-speed shared memory and can be implemented efficiently on GPUs.	With increase in number of queries, the speedup degrades.
Parallel Bloom filters [27]	• First processes elements by allotting them to bucket which is less filled. Then computes 2 hash functions and store the elements based on the bucket it is in. Then transferred to GPU memory. • Key and hash values are calculated and those locations are set. After computation is done data is moved back from GPU to host. The bits positios are XORed to get presence or absence of keys.	GPU and CPU is independent of the key size. As only two unsigned integers are involved in computations for all key sizes.	It Supports insertion or query but not both simultaneously. Also the preprocessing step is very time consuming.

Chapter 4

System Architecture of Velvet Assembler

DNA assembly is a highly computation intensive task which can take up to 4-5 days when done on serial assemblers. To decrease the time of the assembly procedure, multithreading is implemented in a few assemblers. Much work has been done to develop parallel sequence assemblers and parallelize existing serial assemblers. Many existing tool like Velvet supports multithreading through openmp. However there is still a scope of optimization using GPU which is many core and can parallelize the task over thousands of threads. This project work focuses on gaining speedups in velvet using GPU.

4.1 Working of Velvet

Velvet Assembler works in two parts:[15]

- Velveth: Maps k-mers onto hash tables
- Velvetg: create debruijn graph from k-mers and applies error correction

The flowchart gives the Velvet workflow

4.1.1 Velveth Working

Velveth reads k-mers from the DNA reads and hash them to create Roadmaps which in later stage helps in finding path from one read to other. It produces two files Sequences and Roadmaps which functions as input to Velvetg. A typical roadmap construction by velvet consists of following steps:

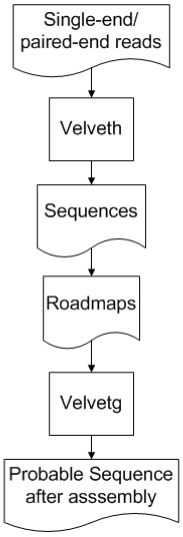

Figure 4.1: Velvet Workflow

Procedure:

1. Velveth scans the reads and store them into Sequences files. This is to store the reads in a format recognizable by velvet assembler. This can include putting paired data from different libraries into a single file.

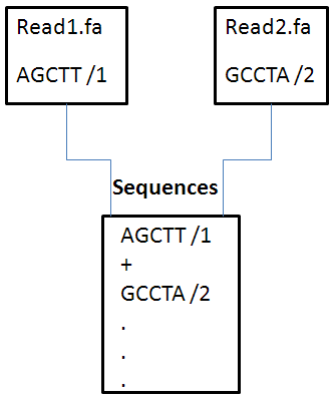

Figure 4.2: Reads to Sequence Conversion

2. String reads can consume too much space therefore they are compressed into a structure TightString. The four nucleotides are represented using two bits each with different combination (A=00, C=01, G=10, T=11). Advantage : Less memory - 40 length read requires 10 bytes instead of 40 bytes and makes Computation faster - Makes hashing efficient.

3. TightString are then divided into overlapping subsequences called k-mers. The kmer

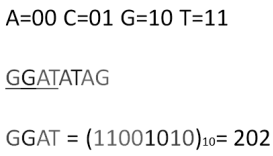

Figure 4.3: TightString creation

size should be optimal, i.e. if k is too small it will produce many misassembles as it is increasing connectivity and a large k will decrease connectivity thus create difficulty in finding path through reads.

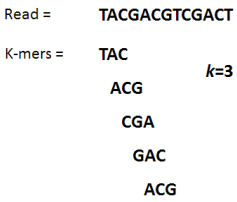

Figure 4.4: k-mers for of k=4

4. The next step is creation of hash-table and checking for the presence of a k-mer in this hash-table. A k-mer value which is referenced for the first time is inserted to the hash table. If already present, it stores a reference of that entry in the Roadmaps file. The hash table and Roadmaps are used by velvetg to assemble the complete DNA.

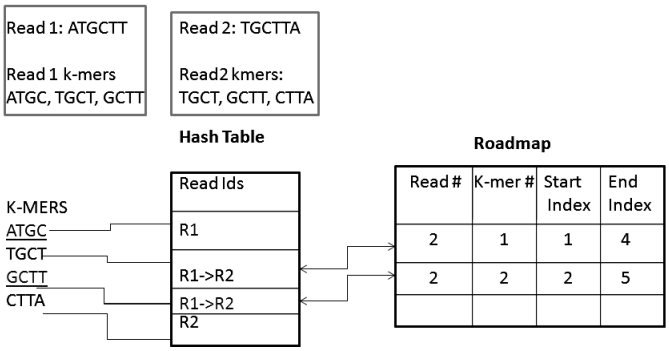

Figure 4.5: Roadmaps creation

4.1.2 Velvetg Working

1. De-Bruijn Graph Construction: Make a vertex for each unique K-mer and a directed edge between two vertices if they overlap by k-1 nucleotides

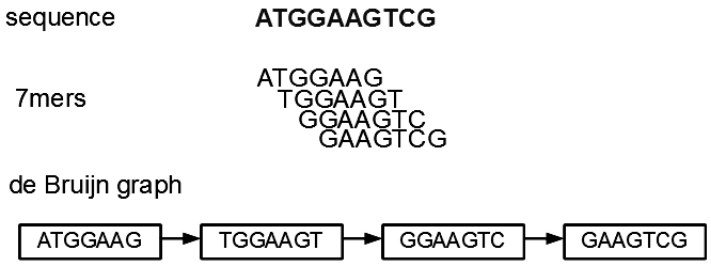

Figure 4.6: De-Bruijn Graph Construction

2. Graph Condensation

   - Node created if there is distinct interruption points.
   - Each node attached to twin node
   - Reverse series of reverse complement k-mers
   - Overlap between reads from opposite strand

3. Simplify chains of blocks

   - Merge if unambiguous path between two nodes
   - Remove errors
   - Reads are traced through the graph

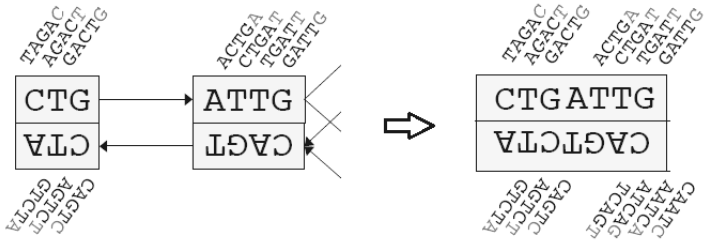

Figure 4.7: Graph Simplification

Chapter 5

Research Gaps and Optimization Scope in Velvet

5.1 Hotspot Identification

Using various profiling tools on velveth we have identified which modules are time consuming, compute intensive and amenable to parallelization. The following chart gives the time taken by different routines of velveth.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 48.73     0.19     0.19   714290     0.00     0.00
findOrInsertOccurenceInSplayTree
 10.26     0.23     0.04        1    40.01    40.01
newTightStringArrayFromStringArray
  7.69     0.26     0.03   142858     0.00     0.00
velvetifySequence
  5.13     0.28     0.03   142858     0.00     0.00
printAnnotations
  3.85     0.30     0.02  5000030     0.00     0.00
getNucleotide
  2.56     0.31     0.02  5000030     0.00     0.00
pushNucleotide
  2.56     0.32     0.01  5000030     0.00     0.00
reversePushNucleotide
  2.56     0.33     0.01  2173788     0.00     0.00
compareKmers
  2.56     0.34     0.01  1714296     0.00     0.00  getLength
```

Figure 5.1: Velveth Profile

Table 5.1: Velveth Profiling

Module	Input/Output Intensive	Computation Intensive
findOrInsertOccurenceInSplayTree	no	yes
newTightStringArrayFromStringArray	no	yes
velvetifySequence	yes	no
printAnnotations	yes	no
getNucleotide	yes	no
pushNucleotide	yes	no
compareKmers	no	yes

5.2 Research gaps and Optimization scope

Research Gap 1: We can see from the profile that function findOrInsertOccurenceInSplayTree (k-mer hashing) is taking most of the time followed by newTightStringArrayFromStringArray(for tightstring construction). Creating tightstring is one of the first steps in velveth and only a single call to it is taking 10% of execution time. No parallelization is provided for newTightStringArrayFromStringArray module.

Research Gap 2: There is redundant processing in conversion of reads file to Sequence file if the reads are single ended reads. For mate pairs having read across multiple files, the Sequence file generated is the collection of ordered pairs. But for single ended reads, the Sequence file is just the rewriting of read file.

Optimization Scope: Function findOrInsertOccurenceInSplayTree for hashing k-mers to create Roadmaps is the most time consuming process in velveth. It is availaible with openmp parallelization but can be further optimized using GPU having many-cores.

Chapter 6

Proposed Work

6.1 Stage 1: Parallelization Using GPU

6.1.1 Heuristics for Hybrid Computing

Parallelizing a sequential code requires analysis of multiple techniques to find which technique is the most optimal choice for parallelization. If the transfer time on GPU alone is greater than the time with multi-threading on CPU then even with low execution time on GPU, GPU is not a suitable choice of HPC tool. Also if the work is Input-Output Intensive, it is not suitable for GPU.

6.1.2 Optimized Parallelization of TightString Construction

Based on the research gaps and optimization scopes identified in the previous chapter we see that TightString construction is highly computational task performing similar instruction execution on data thus making it a good candidate for GPU optimization.

Observation All reads are of similar length, so we allocate memory equal to maximum read length.

Approaches based on Assumption This research focuses on experimenting with different approaches for data distribution and reordering to get maximum efficiency for TightString construction.[29]

Approach 1: Simple Distribution [29] In this approach a single thread processes a complete read. In sequential we can consider that task division is on single thread. By increasing the number of threads we can increase the task being done at a time which will give us low time of execution.

Task Distribution : Whole read per thread

Parallelization achieved = #reads / #threads

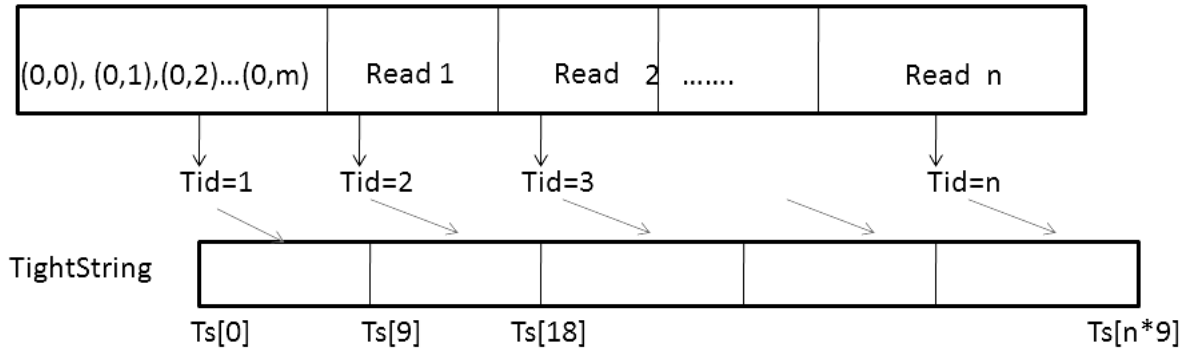

Figure 6.1: Simple distribution

Approach 2: Increased Concurrency [29] In this approach we are increasing the data distribution by processing only four elements of a read by a single thread. This will increase concurrency considerably and will show higher speedups.

Parallelization achieved = 4 times

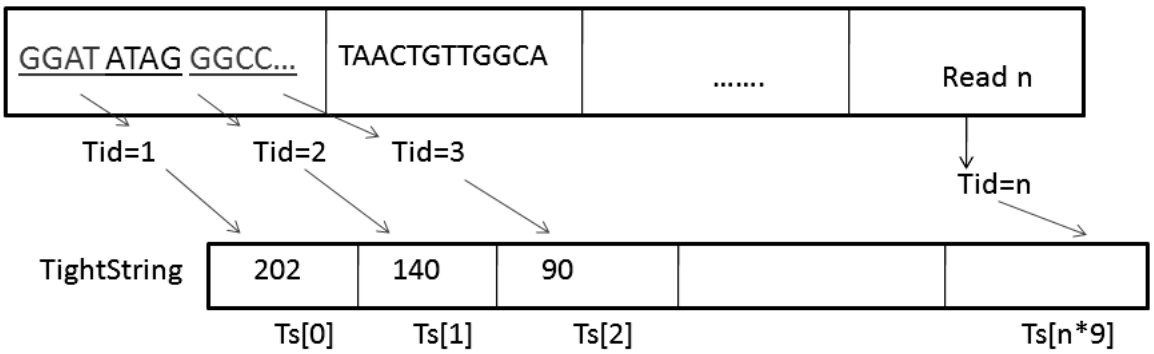

Figure 6.2: Increasing concurrency

Approach 3: Data Reordering [29] In this approach we make use of data reordering to enable coalesced memory access so that the maximum data which is to be processed is loaded into cache at the same time so there is less cache miss. Single thread is processing single read.

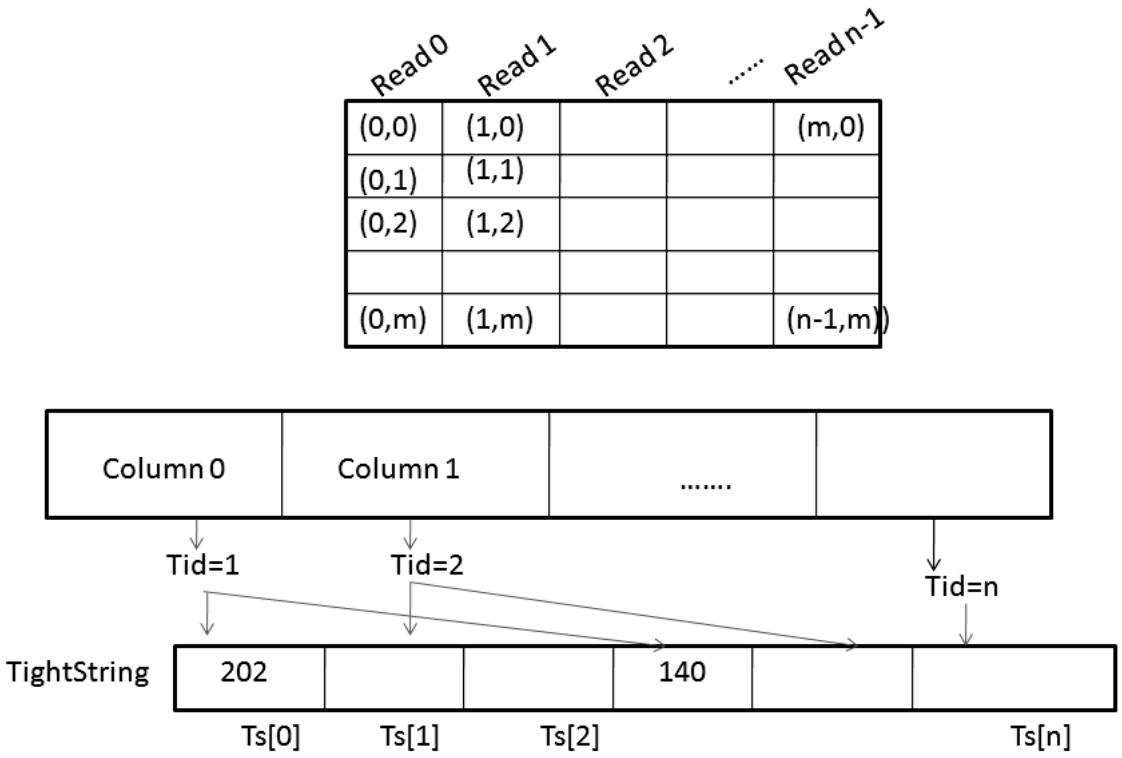

Figure 6.3: Decreasing Cache miss ratio by data reordering

6.1.3 Parallelization of k-mer hashing for Roadmap creation

From the literature surveyed we found that Real time parallel hashing approach modified to suit our data is most suitable. To hash our data we can construct two hash table, table1, the primary table and table2 the secondary. The first table will be of size 225 and second table of size 219. Our hash fuction will be such that most of the keys are mapped on first table without collision. And only in some cases when collision occur the key should be inserted into second hash table which acts as overflow table to accommodate colliding keys. If we find the same kmer in any of the two tables we make a roadmap entry. If it is not present and we have an empty slot available for it we insert it into the table. If both slots are occupied the algorithm evicts the key from slot of first table,

inserts itself and now try to hash the evicted value. Our algorithm should be such that there is no possibility of a key not finding a slot after maximum two iterations. The flowchart gives the flow for roadmap creation using parallel hashing:

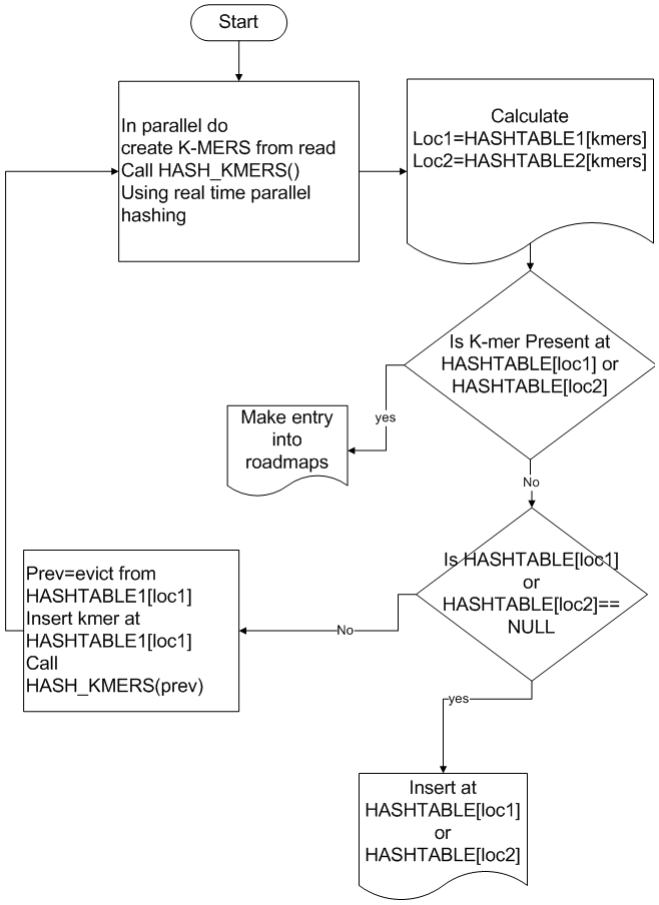

Figure 6.4: Workflow for hashing k-mers

6.2 Stage 2: Architecture Independent Policy Development

To have maximum occupancy of resources, heuristics need to be applied for finding the most optimal block sizes(number of blocks in an SM) based on the size and locality of our data. If we have less number of threads then we will not be able to achieve high concurrency thus wasting our resources. If the block size is high, then in case if our data is significantly smaller than our block size, many threads will be wasted. For minimizing wastage block size should be minimum. So our policy for minimum wastage is setting

the thread size to

thread_size = Maximum threads per SM / Maximum blocks possible per SM

For our data of size 142858*9 in tightstring creation we found out that 192 is the optimal block size which gives maximum occupancy by utilizing maximum possible blocks of SM thus giving efficient performance.

6.3 Stage 3: Redundancy Removal in Sequence file Construction

The Sequence file construction for single end reads is a redundant step as it simply copies data from reads file to Sequences file. To remove this redundancy we propose the following work:

1. if(reads.type==single_end)
2. Use read file as Sequence file
3. else
4. Convert reads to Sequence

Chapter 7

Experiment Setup and Results Analysis

7.1 Experiment Setup

Table 7.1: CPU Configuration

Cores	4
Clock Speed	3.2GHz

Table 7.2: Graphic card Configuration

Card	Nvidia GeForce GTX 480
Compute Capability	2.0
SM	15
Maximum blocks per SM	8
Maximum threads per SM	1536
Maximum threads per Block	1024
Global Memory	1536 MB
Warp Size	32

Operating System

Ubuntu 10.04 or above

Software

Velvet(1.2.10)

Test Data

Test reads provided with Velvet assembler

7.2 Velvet Analysis

7.2.1 Simulation 1 : Finding optimal k-mer size and efficiency of accuracy

Table 7.3: K-mer size vs. N50

K-MER SIZE	CONTIG COUNT	N50	MAX
11	8996	22	132
13	386	807	3343
15	22	16644	20835
17	2	99979	99979
19	1	99977	99977
21	1	99975	99975
23	1	99973	99973
25	2	79560	79650
27	6	20409	37295
29	54	2859	6079
31	436	274	1061

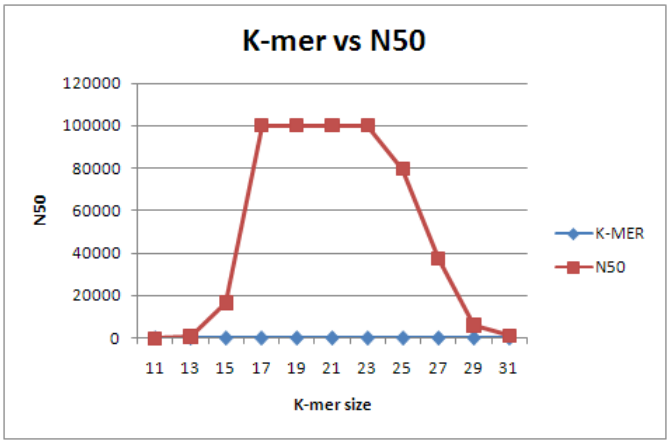

Figure 7.1: K-mer vs. N50 size

Analysis

- As can be seen from the graph the N50 size increases as the kmer size increases as it results in correct long contigs.
- The most optimal value can be found near half the read size. Thus we should not take kmers less than half the read size.
- At too high k-mer size the coverage is low thus producing mis-assemblies.

7.2.2 Simulation 2: Velvet assembly in Multithreaded Mode

Table 7.4: Velvet Assembly in multithreaded mode

Threads	Time for FillTightstring	Time in sec (velveth)
Sequential	28.144	.697
2	30.37	.655
4	32.68	.574
8	33.21	.612

Analysis:

The system being used is quad core system and gives maximum performance at threads=4. Therefore it would be best to set the number of threads to be equal to the number of CPU cores.

7.2.3 Experimental Results for Parallelization of TightString Creation

Table 7.5: Execution time of Parallel vs Sequential

Sequential Implement -ation	Parallel Implementation on GPU								
		Execution Time (ms) on GPU							
	Approach	Without Memory Transfer				With Memory Transfer			
		Block size=128	Block size=192	Block size=256	Block size=512	Block size=128	Block size=192	Block size=256	Block size=512
28.12113 ms	Simple Distribution	1.051103	1.051103	1.04584	0.97521	4.346748	4.32159	4.27259	4.14236
	Increased Concurrency	0.491382	0.48371	0.487725	0.48767	3.65791	3.55643	3.64043	3.56046
	Data Reordering	0.496928	0.46194	0.46686	0.46288	3.6771	3.58473	3.65528	3.613106

From the results we can see that by proper data distribution and reordering we can achieve significant reductions in execution time and the maximum reduction is obsereved for the block size of 192 as determined by the heuristics.

Table 7.6: Speedups of Parallel Approaches

	Speedup = Execution time of Sequential code / Execution time of Parallel code							
Approach	**Without Memory Transfer**				**With Memory Transfer**			
	Block size=128	**Block size=192**	**Block size=256**	**Block size=512**	**Block size=128**	**Block size=192**	**Block size=256**	**Block size=512**
Simple Distribution	26.75392	26.77922	26.88855	28.83597	6.46946	6.50712	6.58175	6.78867
Increased Concurrency	57.22865	58.13634	57.65775	57.6642	7.68775	7.90712	7.72467	7.89817
Data Reordering	56.58994	60.87615	60.2346	60.75252	7.64763	7.8447	7.69329	7.78309

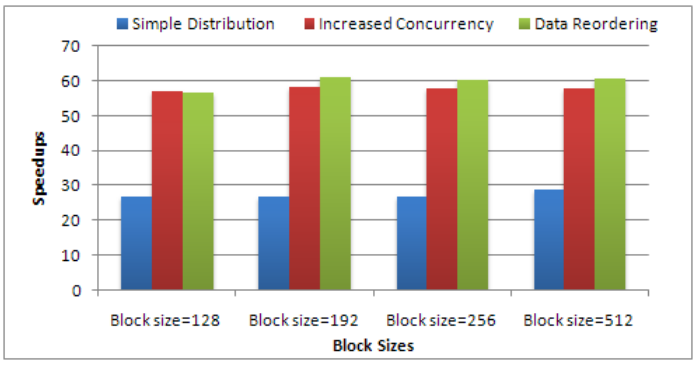

Figure 7.2: Speedup of parallel approaches (Without memory transfer cost)

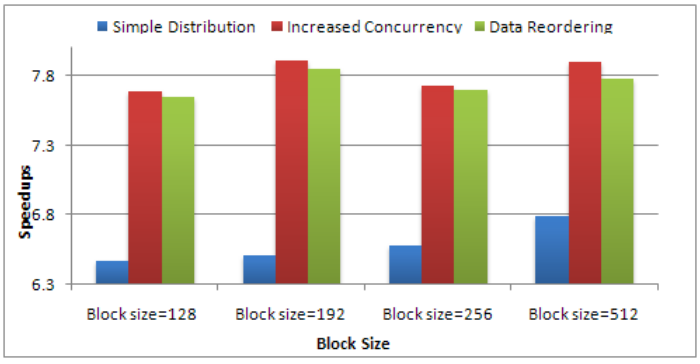

Figure 7.3: Speedup of parallel approaches (With memory transfer cost)

Chapter 8

Conclusion and Future Scope

8.1 Conclusion

Based on results observed by parallelization with GPU on tightstring we can see there is significant speedup in performance with accuracy maintained. We also determined an archtecture independent policy for finding the most optimal block size such that threads wastage was minimum. Parallelizing a sequential code requires analysis of multiple techniques to find which technique is the most optimal choice for parallelization. For example if the transfer time on GPU alone is greater than the time with multi-threading on CPU then even with low execution time with GPU, GPU is not a suitable choice of HPC tool. Therefore parallelizing a sequential code should be done while keeping in mind various heuristics.

8.2 Future Scope

For the future work we can work on parallelizing the Roadmap creation which is the most time consuming process of velveth. Also we can use heuristic based methods to reduce redundancy in Sequences construction using Cluster as it is an input output intensive task.

Bibliography

[1] NVIDIA Corporation, *NVIDIA CUDA C Programming Guide*, June 2011.

[2] F. Sanger, S. Nicklen, and A. Coulson, "DNA sequencing with chain-terminating inhibitors," *Proceedings of The National Academy of Sciences of The United States Of America*, vol. 74, 1977.

[3] J. R. Miller, S. Koren, and G. Sutton, "Assembly algorithms for next-generation sequencing data," *Genomics*, vol. 95, no. 6, pp. 315 – 327, 2010.

[4] C. T. C. A. Edwards, "Closure strategies for random dna sequencing."

[5] M. S. Rosenberg, *Sequence Alignment Methods, Models, Concepts, and Strategies*. University of California Press (August 22, 2011), 2011.

[6] "Sequence alignment," http://en.wikipedia.org/wiki/Sequence_alignment.

[7] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403–410, 1990.

[8] I. Lobo, "Basic local alignment search tool (blast)," *Nature Education*, 2008.

[9] J. C. Venter, ""shotgunning the human genome: A personal view. encyclopedia" of life sciences, 2006."

[10] Samuel Myllykangas , Jason Buenrostro , and Hanlee P. Ji, "Overview of sequencing technology platforms."

[11] Y. Liu, B. Schmidt, and D. Maskell, "DecGPU: distributed error correction on massively parallel graphics processing units using CUDA and MPI," *BMC Bioinformatics*, vol. 12, 2011. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-12-85

[12] R. M. Idury and M. S. Waterman, "A new algorithm for dna sequence assembly." *Journal of Computational Biology*, 1995.

[13] M. A. Alekseyev and P. A. Pevzner, "Colored de bruijn graphs and the genome halving problem." *IEEE/ACM Trans. Comput. Biology Bioinform.*, vol. 4, 2007.

[14] P. A. Pevzner, H. Tang, and M. S. Waterman, "An eulerian path approach to dna fragment assembly," *Proceedings of the National Academy of Sciences*, vol. 98, no. 17, pp. 9748–9753, 2001.

[15] E. B. D. Zerbino, "Velvet: algorithms for de novo short read assembly using de bruijn graphs," *Genome Research*, no. 5, pp. 821–829, 2008.

[16] J. Simpson, K. Wong, S. Jackman, J. Schein, S. Jones, and I. Birol, "Abyss: a parallel assembler for short read sequence data." *Genome Res*, vol. 19, p. 1117, 2009.

[17] J. Butler, I. MacCallum, M. Kleber, I. Shlyakhter, M. Belmonte, E. Lander, C. Nusbaum, and D. Jaffe, "Allpaths: de novo assembly of whole-genome shotgun microreads." *Genome Res*, vol. 18, 2008.

[18] T. Gingeras, J. Milazzo, D. Sciaky, R. Roberts, "Computer programs for the assembly of dna sequences," *Nucleic Acids Research, vol. 7, no. 2, pp. 529543*, 1979.

[19] "Denovo assembly using illumina short reads," Illumina Inc, 2010.

[20] Y. Liu, B. Schmidt, and D. Maskell, "Parallelized short read assembly of large genomes using de Bruijn graphs," *BMC Bioinformatics*, vol. 12, 2011.

[21] S. Boisvert, F. Raymond, E. Godzaridis, F. Laviolette, and J. Corbeil, "Ray Meta: scalable de novo metagenome assembly and profiling." *Genome Biol.*, vol. 12 [Epub ahead of print], 2012.

[22] X. Liu, P. R. Pande, H. Meyerhenke, and D. A. Bader, "Pasqual: Parallel techniques for next generation genome sequence assembly," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 5, pp. 977–986, 2013.

[23] S. Lefebvre and H. Hoppe, "Perfect spatial hashing." *ACM Trans. Graph.*, vol. 25, 2006. [Online]. Available: http://dblp.uni-trier.de/db/journals/tog/tog25.html#LefebvreH06a

[24] R. Pagh and F. F. Rodler, "Cuckoo hashing," *J. Algorithms*, May 2004.

[25] I. García, S. Lefebvre, S. Hornus, and A. Lasram, "Coherent parallel hashing," *ACM Trans. Graph.*, Dec. 2011.

[26] J. Pan and D. Manocha, "Fast gpu-based locality sensitive hashing for k-nearest neighbor computation," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '11. New York, NY, USA: ACM, 2011, pp. 211–220. [Online]. Available: http://doi.acm.org/10.1145/2093973.2094002

[27] P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison, "Scalable bloom filters," *Inf. Process. Lett.*, vol. 101, no. 6, pp. 255–261, Mar. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.ipl.2006.10.007

[28] D. A. Alcantara, A. Sharf, F. Abbasinejad, S. Sengupta, M. Mitzenmacher, J. D. Owens, and N. Amenta, "Real-time parallel hashing on the gpu," *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2009)*, Dec. 2009.

[29] V. Shah, M. ; Patel, "An efficient sparse matrix multiplication for skewed matrix on gpu."