GPU-Accelerated Large-Scale Genome Assembly

Minh Le & Chile Asante

Abstract

Traditional novel genome sequencing has been leveraging the power of CPU-based application to handle assembly workload, which is both compute-intensive and data-intensive. Such workloads require large hard drive space, usually up to several hundreds of gigabytes of real sequence datasets, thus prevent the process to be leveraged by GPU-based application, which could only handle up to tens of gigabytes of data storage.

In this paper, the authors introduced LaSAGNA, a new GPU-accelerated genome assembler that uses a semi-streaming approach based on the available memory. They also proposed a two-level streaming approach to minimize disk Input/Output.

Genome Sequencing and Assembly

Sequencing: the process of determining the exact order of nucleotides (production unit of 2 essential biomolecules within all life-forms) in the DNA.

Existing technology cannot read a whole genome (a complete set of DNA of an organism, including all of its genes - which is 3 billion DNA base pairs) in one go. Therefore, scientists clone it and extract millions of smaller fragments (called *short-reads*). Assembly is the process of aligning those short-reads into the original sequence.

Recent Advancements and Challenges

The Illunina HiSeq 4000 sequencer can generate up to 5 billion 150-nucleotide bases at the cost of 5 cents per million bases.

The human genome set can size up to half a Terabyte, which would require a machine with terabytes of RAM or scale-out cluster with dozens of nodes, which are not available commercially and often too expensive for most researchers.

Traditionally, applications have been utilizing the processing power of CPU chips, however, recently researchers have realized the power of GPU cards, which have a much higher processing capability.

Benefits of using GPUs

Speed: recent NVIDIA Tesla V100 has a theoretical performance of 15 TFLOP/s of single precisions and 900GB/s of peak bandwidth memory, compared to 85GB/s the latest Intel Xeon Processor.

LaSAGNA can build an approximate overlap graph from a real-world genome sequence (hundreds of GBs in size) using only 1 GPU with 6GB of memory (in 17 hours)

Distributed version of LaSAGNA that utilizes a cluster of nodes to distribute computations and increase In/Out throughput.

Methodology for GPU-Accelerated Assembly

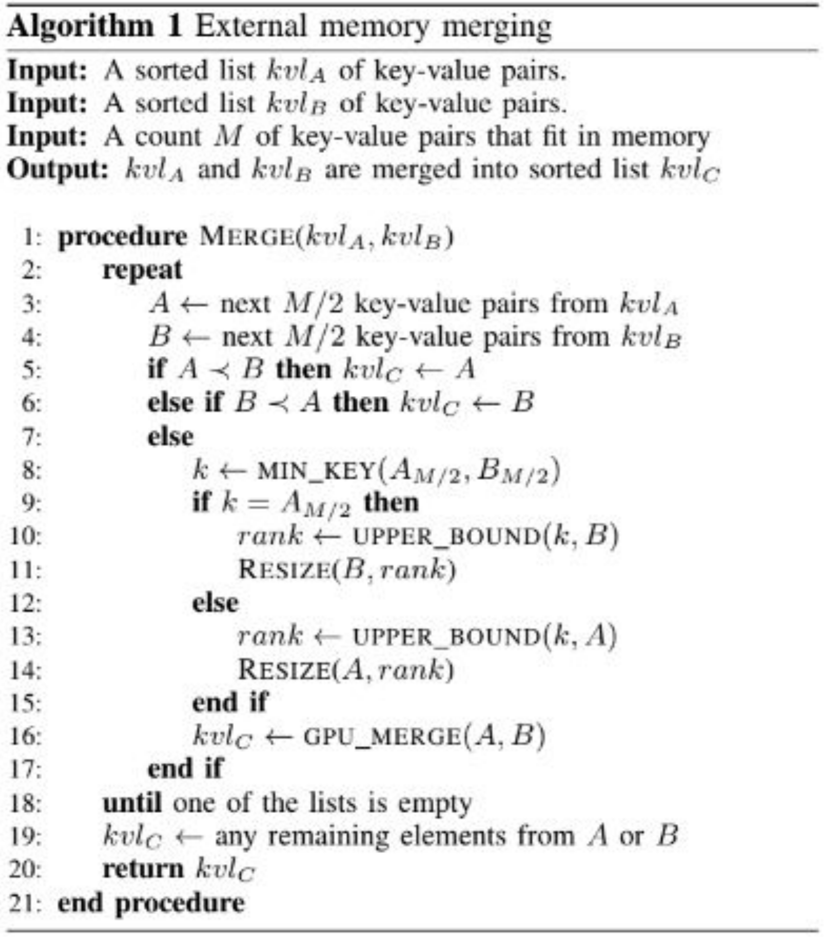
- Map
    - Generate pairs of fingerprints and read-IDs
    - Reads are loaded into GPU
    - Fingerprint generation is parallelized by assigning each read to a thread
    - Does not utilize shared memory
- Sort
    - Sort read-IDs by fingerprints
    - Uses an external-memory sorting scheme comprising two phases
        - First phase
            - Chunks of key value pairs are read from disk, sorted by keys, and written back to the disk
        - Second Phase
            - Chunks are merged into a single sorted one

GPU Methodology cont.

- Reduce
    - Find suffix-prefix matches
    - Consumes two lists S_1 and P_1 containing tuples of the read-IDs and fingerprints
    - For each match the system adds an edge to the string graph for each matching read-ID.
    - This is basically another sorting algorithm
- Compress
    - Traverse paths and generate contigs
    - This phase consists of two stages.
        - First stage
            - Traverse the string graph to obtain a set of paths
        - Second stage
            - Convert read-IDs belonging to the paths in the string graph to their corresponding input sequences to generate contiguous sections (contigs) of the original DNA sequence.

Sort: Algorithm 1

Algorithm 1 explains our approach to combine two sorted lists on disk, by processing at least M/2 and at most M elements at a time. It is adapted from the *k-way* merging scheme and operates under the guiding principle that the input cannot be randomly accessed.



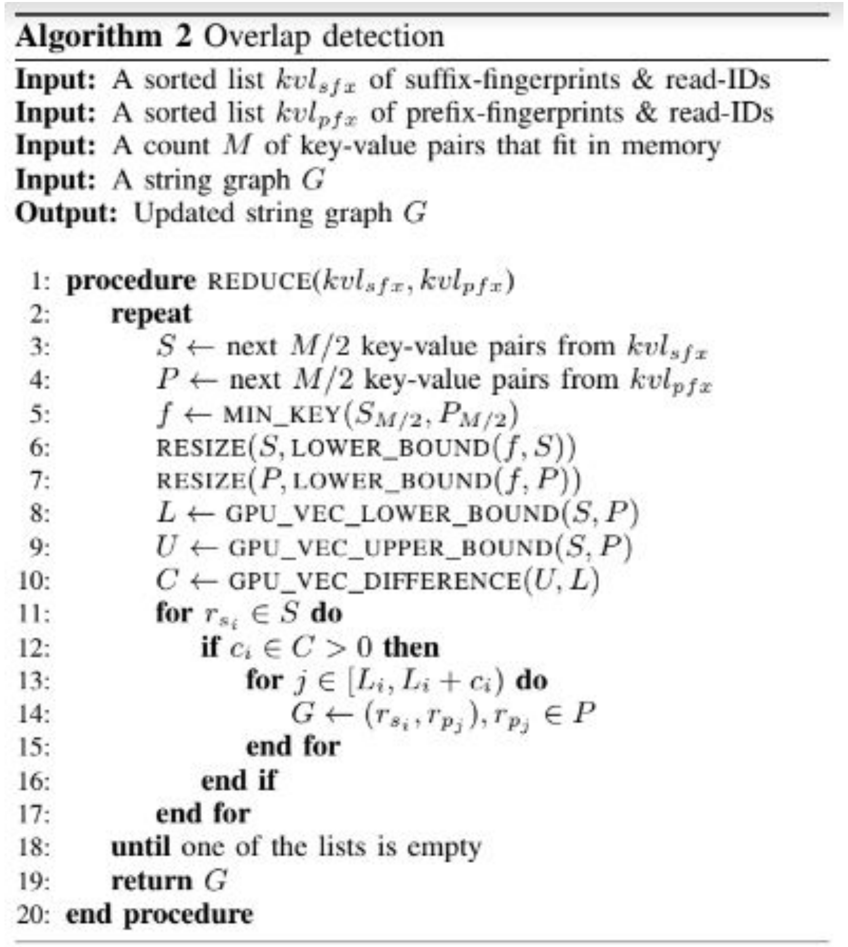
Algorithm 1 External memory merging

Input: A sorted list kvl_A of key-value pairs.
Input: A sorted list kvl_B of key-value pairs.
Input: A count M of key-value pairs that fit in memory
Output: kvl_A and kvl_B are merged into sorted list kvl_C

```
procedure MERGE($kvl_A, kvl_B$)
    repeat
        $A \leftarrow$ next $M/2$ key-value pairs from $kvl_A$
        $B \leftarrow$ next $M/2$ key-value pairs from $kvl_B$
        if $A \prec B$ then $kvl_C \leftarrow A$
        else if $B \prec A$ then $kvl_C \leftarrow B$
        else
            $k \leftarrow$ MIN_KEY($A_{M/2}, B_{M/2}$)
            if $k = A_{M/2}$ then
                $rank \leftarrow$ UPPER_BOUND($k, B$)
                RESIZE($B, rank$)
            else
                $rank \leftarrow$ UPPER_BOUND($k, A$)
                RESIZE($A, rank$)
            end if
            $kvl_C \leftarrow$ GPU_MERGE($A, B$)
        end if
    until one of the lists is empty
    $kvl_C \leftarrow$ any remaining elements from $A$ or $B$
    return $kvl_C$
end procedure
```

Reduce: Algorithm 2

- Streams data from the list of suffixes and prefixes into host memory with at most M/2 key-value pairs per window (lines 3 and 4).
- Finds the smaller of the largest fingerprints from either window (f) and calculate the lower-bound of f in both the windows.



Algorithm 2 Overlap detection

Input: A sorted list kvl_{sfx} of suffix-fingerprints & read-IDs
Input: A sorted list kvl_{pfx} of prefix-fingerprints & read-IDs
Input: A count M of key-value pairs that fit in memory
Input: A string graph G
Output: Updated string graph G

```
 1: procedure REDUCE($kvl_{sfx}, kvl_{pfx}$)
 2:     repeat
 3:         $S \leftarrow$ next $M/2$ key-value pairs from $kvl_{sfx}$
 4:         $P \leftarrow$ next $M/2$ key-value pairs from $kvl_{pfx}$
 5:         $f \leftarrow$ MIN_KEY($S_{M/2}, P_{M/2}$)
 6:         RESIZE($S$, LOWER_BOUND($f, S$))
 7:         RESIZE($P$, LOWER_BOUND($f, P$))
 8:         $L \leftarrow$ GPU_VEC_LOWER_BOUND($S, P$)
 9:         $U \leftarrow$ GPU_VEC_UPPER_BOUND($S, P$)
10:         $C \leftarrow$ GPU_VEC_DIFFERENCE($U, L$)
11:         for $r_{s_i} \in S$ do
12:             if $c_i \in C > 0$ then
13:                 for $j \in [L_i, L_i + c_i)$ do
14:                     $G \leftarrow (r_{s_i}, r_{p_j}), r_{p_j} \in P$
15:                 end for
16:             end if
17:         end for
18:     until one of the lists is empty
19:     return $G$
20: end procedure
```

What is the main purpose of the Reduce Algorithm?

A. Find suffix-prefix matches
B. Traverse paths and generate contigs
C. Generate pairs of fingerprints and read-IDs
D. Sort read-IDs by fingerprints
E. None of the Above

What is the main purpose of the Reduce Algorithm?

A. **Find suffix-prefix matches**
B. Traverse paths and generate contigs
C. Generate pairs of fingerprints and read-IDs
D. Sort read-IDs by fingerprints
E. None of the Above

Why does GPU Mapping fail to perform as expected?

A. Rabin Karp’s rolling hash
B. Overlapping pairs of reads
C. Short-reads are too large to fit in thread IDs
D. GPU system mapping is not effective in this experiment
E. Number of threads block memory access

Why does GPU Mapping fail to perform as expected?

A. Rabin Karp’s rolling hash
B. Overlapping pairs of reads
C. Short-reads are too large to fit in thread IDs
D. GPU system mapping is not effective in this experiment
E. **Number of threads block memory access**

Excessive memory throttling, a scenario where threads block because of numerous pending memory accesses. This scheme does not utilize shared memory.

Compress: Traverse Paths and Generate Contigs

2 stages:

First stage: traverse the string graph to obtain a set of paths (each path is a sequence of tuples- each one contains a read-ID and its overhang-length. Traversal starts with vertices with in-degree 0 and out-degree 1 as seeds. From each seed, extend the path by appending the read-ID and overhang length of the current vertex to the sequence of tuples, and stop when found a vertex with no outgoing edges.

Second stage: convert read-IDs belonging to the paths in the graph to their corresponding sequences to generate contigs of the original DNA sequence.

Distributed Graph Over Multiple Compute Nodes

Most prominent bottleneck in the pipeline is the Input/Output throughput. Distribute the computation across multiple nodes to achieve higher bandwidth.

Map: each node requests the master for the address of an input block, then processes the sequences in the blocks, generate tuples, and write on to local disk. The tuples are split into partitions based on the length of suffixes and prefixes, each would remain in a separate file.

Shuffle and sort: each node works on separate partitions and aggregate the data assigned to it from other peers before sorting.

Reduce: partition pairs by the order of their lengths, a node reducing partition p(i) must wait for the node reducing p(i+1) to finish.

Evaluation Tables for LaSAGNA

- Table II and Table III show the total assembly times with details of each phase for different datasets
- The execution times increase only slightly for all other datasets except H.Genome.
- Run-times of the other phases are similar because LaSAGNA performs the same amount of I/O

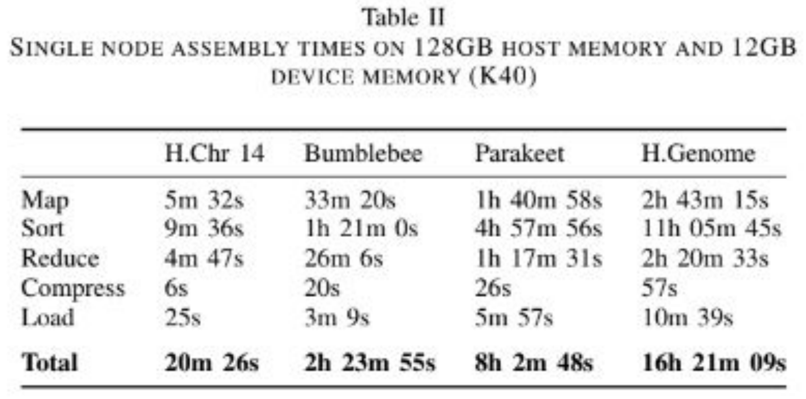


Table II
SINGLE NODE ASSEMBLY TIMES ON 128GB HOST MEMORY AND 12GB DEVICE MEMORY (K40)

	H.Chr 14	Bumblebee	Parakeet	H.Genome
Map	5m 32s	33m 20s	1h 40m 58s	2h 43m 15s
Sort	9m 36s	1h 21m 0s	4h 57m 56s	11h 05m 45s
Reduce	4m 47s	26m 6s	1h 17m 31s	2h 20m 33s
Compress	6s	20s	26s	57s
Load	25s	3m 9s	5m 57s	10m 39s
Total	**20m 26s**	**2h 23m 55s**	**8h 2m 48s**	**16h 21m 09s**

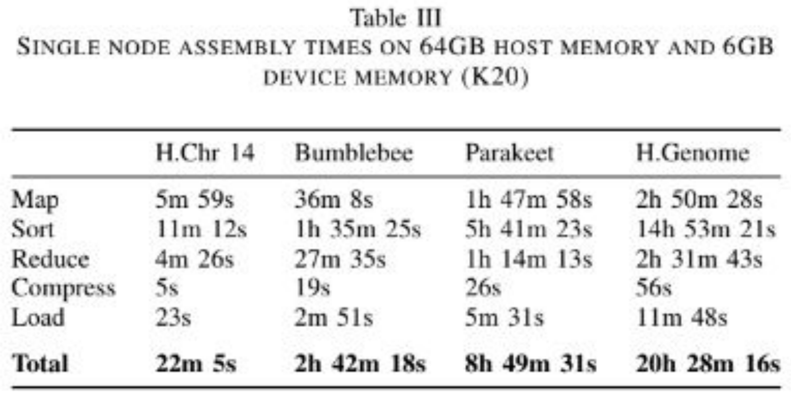


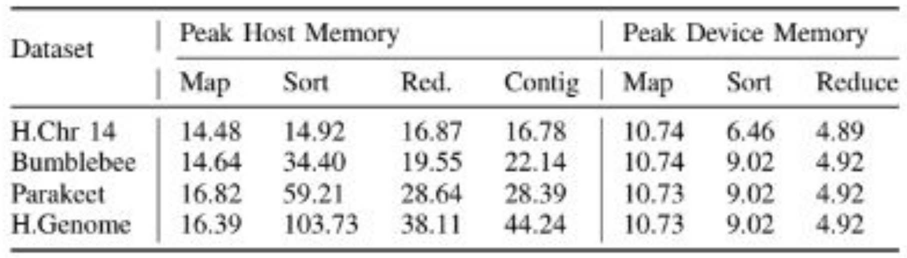
Table III
SINGLE NODE ASSEMBLY TIMES ON 64GB HOST MEMORY AND 6GB DEVICE MEMORY (K20)

	H.Chr 14	Bumblebee	Parakeet	H.Genome
Map	5m 59s	36m 8s	1h 47m 58s	2h 50m 28s
Sort	11m 12s	1h 35m 25s	5h 41m 23s	14h 53m 21s
Reduce	4m 26s	27m 35s	1h 14m 13s	2h 31m 43s
Compress	5s	19s	26s	56s
Load	23s	2m 51s	5m 31s	11m 48s
Total	**22m 5s**	**2h 42m 18s**	**8h 49m 31s**	**20h 28m 16s**

Evaluation Tables cont.

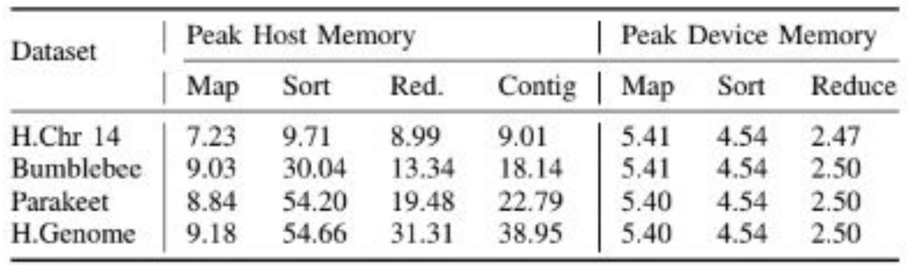
- Table IV shows the peak host and device memory usage during the various phases of assembly for different datasets
- Table V shows the results when run on a SuperMic node with 64 GB host memory and one NVIDIA K20 with 6 GB device memory
- In both cases, the device memory usage is almost identical for all datasets because a fixed amount of device memory is allocated for each phase regardless of the data size

Table IV
PEAK MEMORY USAGE (IN GB) ON 128 GB HOST WITH K40



Dataset	Peak Host Memory				Peak Device Memory		
	Map	Sort	Red.	Contig	Map	Sort	Reduce
H.Chr 14	14.48	14.92	16.87	16.78	10.74	6.46	4.89
Bumblebee	14.64	34.40	19.55	22.14	10.74	9.02	4.92
Parakeet	16.82	59.21	28.64	28.39	10.73	9.02	4.92
H.Genome	16.39	103.73	38.11	44.24	10.73	9.02	4.92

Table V
PEAK MEMORY USAGE (IN GB) ON 64 GB HOST WITH K20



Dataset	Peak Host Memory				Peak Device Memory		
	Map	Sort	Red.	Contig	Map	Sort	Reduce
H.Chr 14	7.23	9.71	8.99	9.01	5.41	4.54	2.47
Bumblebee	9.03	30.04	13.34	18.14	5.41	4.54	2.50
Parakeet	8.84	54.20	19.48	22.79	5.40	4.54	2.50
H.Genome	9.18	54.66	31.31	38.95	5.40	4.54	2.50

Which dataset had the shortest execution time?

A. H.Chr 14
B. Bumblebee
C. Parakeet
D. H. Genome
E. None of the Above

Which dataset had the shortest execution time?

A. **H.Chr 14**
B. Bumblebee
C. Parakeet
D. H. Genome
E. None of the Above

Run-times are similar (except Human Genome) because LaSAGNA performs the same amount of I/O

Which computing environment was used to evaluate LaSAGNA?

A. QueenBee II4
B. NVIDIA Tesla K40 GPUs
C. IBM G63 Dual-Processors
D. E5-2680v2 Xeon processors
E. A few of the above

Which computing environment was used to evaluate LaSAGNA?

A. **QueenBee II4**
B. **NVIDIA Tesla K40 GPUs**
C. IBM G63 Dual-Processors
D. **E5-2680v2 Xeon processors**
E. **A few of the above**

Conclusion

- LaSAGNA
  - a new GPU-accelerated genome assembler, that can assemble large-scale sequence datasets on a single GPU by constructing string graphs from approximate overlaps using fingerprints.
  - Uses a two-level semi-streaming model that exploits the speed of GPU device memory as well as the large capacity of host memory.
  - the first GPU-based assembler that can assemble a real human genome dataset on a single node.

Experimental results demonstrate that LaSAGNA can assemble a 400 GB human genome dataset in 17 hours using a single GPU