CRiSPy-CUDA: Computing Species Richness in 16S rRNA Pyrosequencing Datasets with CUDA

Zejun Zheng[1], Thuy-Diem Nguyen[1], and Bertil Schmidt[2]

[1] School of Computer Engineering, Nanyang Technological University, Singapore
{zjzheng,thuy1}@ntu.edu.sg
[2] Institut für Informatik, Johannes Gutenberg University Mainz, Germany
{bertil.schmidt}@uni-mainz.de

Abstract. Pyrosequencing technologies are frequently used for sequencing the 16S rRNA marker gene for metagenomic studies of microbial communities. Computing a pairwise genetic distance matrix from the produced reads is an important but highly time consuming task. In this paper, we present a parallelized tool (called CRiSPy) for scalable pairwise genetic distance matrix computation and clustering that is based on the processing pipeline of the popular ESPRIT software package. To achieve high computational efficiency, we have designed massively parallel CUDA algorithms for pairwise k-mer distance and pairwise genetic distance computation. We have also implemented a memory-efficient sparse matrix clustering program to process the distance matrix. On a single-GPU, CRiSPy achieves speedups of around two orders of magnitude compared to the sequential ESPRIT program for both the time-consuming pairwise genetic distance module and the whole processing pipeline, thus making CRiSPy particularly suitable for high-throughput microbial studies.

Keywords: Metagenomics, Pyrosequencing, Alignment, CUDA, MPI.

1 Introduction

Pyrosequencing technologies are frequently used for microbial community studies based on sequencing of hyper-variable regions of the 16S rRNA marker gene. Examples include profiling of microbial communities in seawater [1] and human gut [2]. The produced datasets contain reads of average length between 200 and 600 base-pairs. Typical dataset sizes range between a few tens of thousand up to around a million reads. Computational analysis of these datasets can be classified into two approaches: taxonomy-dependent and taxonomy-independent [3,4]. The taxonomy-dependent approach compares the input data against a reference database to assign each read to an organism based on the reported matches. The drawback of this approach is that existing databases are incomplete since the vast majority of microbes are still unknown.

The taxonomy-independent approach performs a hierarchical clustering and then bins the reads into OTUs (Operational Taxonomic Units) based on a distance threshold. Clustering is typically computed on a pairwise genetic distance

M. Loog et al. (Eds.): PRIB 2011, LNBI 7036, pp. 37–49, 2011.

matrix derived from an all-against-all read comparison. The advantage of this approach is its ability to characterize novel microbes. However, the all-against-all comparison is highly compute-intensive. Furthermore, due to advances in pyrosequencing technologies, the availability and size of input read datasets is increasing rapidly. Thus, finding fast and scalable solutions is of high importance to research in this area.

Existing methods for the taxonomy-independent approach can be classified into four categories:

1. multiple sequence alignment (MSA), e.g. MUSCLE [5]
2. profile-based multiple sequence alignment (PMSA), e.g. RDP-aligner [6]
3. pairwise global alignment (PGA), e.g. ESPRIT [7]
4. greedy hierarchical clustering (GHC), e.g. UCLUST [8]

A number of recent performance evaluations [9,10] have shown that both the MSA and PMSA approaches often lead to less accurate genetic distance matrix values than the PGA approach. The GHC approach is faster than the other approaches but produced clusters which are generally of lower quality [10].The main drawback of the PGA approach is its high computational complexity. For an input dataset containing n reads of average length l, the time complexity of all optimal global pairwise alignments is $O(n^2l^2)$. Thus, the application of PGA for metagenomic studies has so far been limited to a relatively small dataset size.

In this paper, we present an approach to extend the application of PGA to bigger datasets. We address the scalability problem by designing a fast solution for large-scale pairwise genetic distance matrix computations using commonly available massively parallel graphics hardware (GPUs) with the CUDA (Compute Unified Device Architecture) programming language. Recent works on using CUDA for fast biological sequences analysis [11,12] have motivated the use of CUDA for pairwise distance matrix computation. The presented tool, called CRiSPy, is based on the popular ESPRIT software [7], which is used by more than 150 major research institutes worldwide. Compared to sequential ESPRIT, CRiSPy provides up to two orders of magnitude speedup for both the pairwise alignment computation and the complete processing pipeline on a single-GPU. We achieve this acceleration by designing efficient massively parallel algorithms of the pairwise k-mer distance and genetic distance matrix computation and by implementing a more scalable hierarchical clustering module.

2 PGA Approach and ESPRIT

We consider an input dataset $R = \{R_1, \ldots, R_n\}$ consisting of n reads over the DNA alphabet $\Sigma = \{A, C, G, T\}$. Let the length of R_i be denoted as l_i and the average length of all reads be l. The PGA approach consists of three steps:

1. Computation of a symmetric matrix D of size $n \times n$, where $D_{i,j}$ is the genetic distance between two reads R_i and R_j
2. Hierarchical clustering of D
3. Using the dendrogram, group reads into non-overlapping OTUs at each given distance level d (e.g. output five OTU grouping for $d = 0.02, 0.03, 0.05, 0.1, 0.2$)

In the PGA approach, the genetic distance $D_{i,j}$ of the two reads R_i and R_j of length l_i and l_j is usually defined as $D_{i,j} = ml/al$, where ml denotes the number mismatches, including gaps (but ignoring end-gaps), in the optimal global alignment of R_i and R_j with respect to a given scoring system and al is the alignment length. The optimal global alignment of R_i and R_j can be computed with the dynamic programming (DP) based NW algorithm [13].

The values of ml and al can be found during the traceback procedure. If all genetic distances are computed using the NW algorithm, the overall amount of DP cells to be calculated is around $3l^2n^2/2$. Assuming input data sizes of $n = 250,000$ and $l = 400$, as well as a computing power of 10 GCUPS (Giga Cell Updates per Second), this procedure would take more than seventeen days. Furthermore, storing the genetic distance matrix would require 116.4 GBytes (using 4 bytes per entry) of memory.

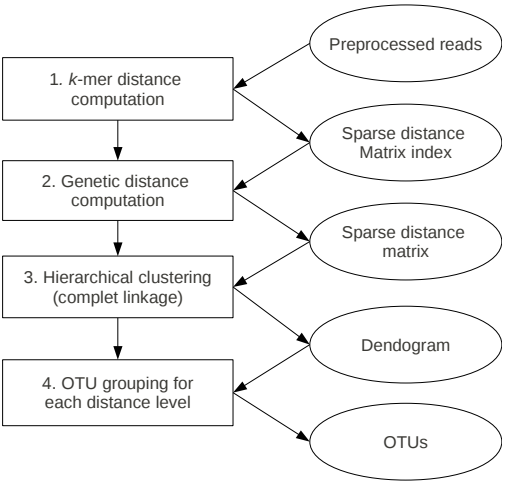

Fig. 1. The flowchart of the ESPRIT algorithm

ESPRIT [7] uses two techniques to reduce runtime and memory requirements: *1.Filtration.* ESPRIT only computes genetic distances for read pairs which have corresponding k-mer distances below a given threshold θ_k. Given two reads R_i and R_j of length l_i and l_j and a positive integer k, their k-mer distance is defined as:

$$d_k(R_i, R_j) = 1 - \frac{\sum_{p=1}^{|\Omega|} \min(n_i[p], n_j[p])}{\min(l_i, l_j) - k + 1}$$

where Ω is the set of all substrings over Σ of length k enumerated in lexicographically sorted order and $n_i(p)$ and $n_j(p)$ are the numbers of occurrences of substring number p in R_i and R_j respectively. This approach is efficient since computation of all pairwise k-mer distances can be done in time $O(ln^2)$. It also relies on the assumptions that k-mer distance and genetic distance are correlated [5] and that a read pair with a large k-mer distance is usually not grouped

into the same OTU. Lower values for θ_k would increase filtration efficiency but decrease sensitivity.

2.Sparse matrix representation. Filtration typically eliminates the majority of read pairs from further consideration. Thus, the k-mer distance matrix and the genetic distance matrix can both be efficiently stored in a sparse matrix format, which reduces memory requirements. Figure 1 shows the processing pipeline of the ESPRIT algorithm.

3 CRiSPy

3.1 Parallel *k*-mer Distance Computation

Although the computation of k-mer distances is around two orders of magnitude faster than the computation of genetic distances, it can still require a significant amount of time. Therefore, we have designed a sorting-based k-mer distance calculation method which can be efficiently parallelized with CUDA.

Initially, a so-called value array V_i is pre-computed for each input read R_i. It consists of all substrings of R_i of length k sorted in lexicographical order. For a pairwise k-mer distance, the two corresponding value arrays are scanned in ascending order. In each step, two elements are compared. If the two compared elements are equal, the indices to both value arrays are incremented and the corresponding counter $min(n_1(i), n_2(i))$ is increased by one. Otherwise, only the index pointing to the value array of the smaller element is incremented. The pairwise comparison stops when the end of one value array is reached. The sorting-based algorithm is illustrated in Figure 2. Obviously, it requires time and space $O(l)$ for two reads of length l each.

```
count:=0; i:=0; j:=0;
while (i < l1 – k + 1) and (j < l2 – k + 1) do
        if V1(i) < V2(j) then
                i++;
        elseif V1(i) > V2(j) then
                j++;
        else
                count++;  i++; j++;
        endif
endwhile
distance:=1 – count / (min(l1,l2) – k + 1);
```

Fig. 2. Sorting-based k-mer distance calculation for two reads R_1, R_2 of length l_1, l_2

The pair indices with k-mer distance smaller than a given threshold value are kept in a sparse index matrix for the subsequent processing stage. In this paper, we use the threshold $\theta_k = 0.3$ for large datasets and $\theta_k = 0.5$ for medium datasets.

3.2 Parallel Genetic Distance Computation

CRiSPy uses a simplified formula for global alignment with affine gap penalty based on the Needleman-Wunsch (NW) algorithm [13]. The scoring matrix of the alignment is computed using the following formula:

$$M(p,q) = \max \begin{cases} M(p-1,q-1) + sbt(R_i[p], R_j[q]) \\ M(p,q-1) + \alpha D(p,q-1) + \beta U(p,q-1) \\ M(p-1,q) + \alpha D(p-1,q) + \beta L(p-1,q) \end{cases}$$

where D, L and U are binary DP matrices to indicate which neighbor (diagonal, left or up) the maximum in cell $M(p,q)$ is derived from. Matrices D, L and U are defined as follows:

$$U(p,q) = 0, L(p,q) = 0, D(p,q) = 1 \; if M(p,q) = M(p-1,q-1) + sbt(R_i[p], R_j[q])$$
$$U(p,q) = 0, L(p,q) = 1, D(p,q) = 0 \; if M(p,q) = M(p,q-1) + \alpha D(p,q-1) + \beta U(p,q-1)$$
$$U(p,q) = 1, L(p,q) = 0, D(p,q) = 0 \; if M(p,q) = M(p-1,q) + \alpha D(p-1,q) + \beta L(p-1,q)$$

Note that $D(p,q) + L(p,q) + U(p,q) = 1$ for $p = 0, \ldots, l_i, q = 0, \ldots, l_j$.

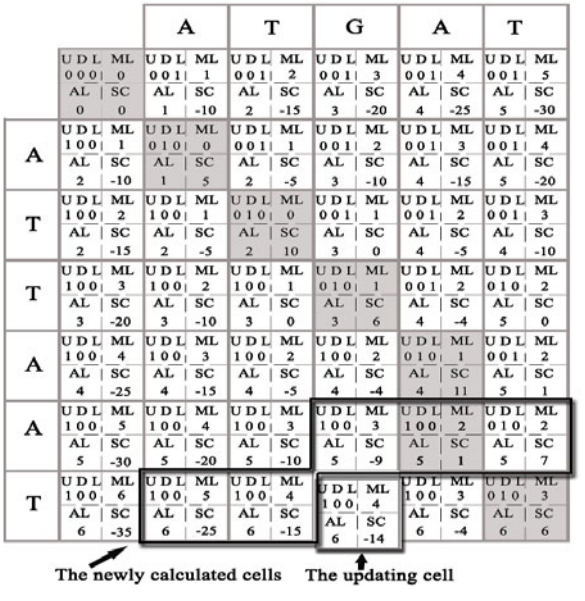

Fig. 3. DP matrices for two input reads ATGAT and ATTAAT with the scoring scheme: $sbt(x = y) = 5$, $sbt(x \neq y) = -4$, $\alpha = -10$, $\beta = -5$

To make the genetic distance calculation more suitable for parallelization, we have designed a trace-back-free linear space implementation by merging the *ml* and *al* calculation into the DP computation of the optimal global alignment score. To obtain the values *ml* and *al*, we introduce two more matrices ML and AL with the recurrent relations as follows:

$$ML(p,q) = U(p,q)ML(p,q-1) + L(p,q)ML(p-1,q) + D(p,q)ML(p-1,q-1) - m(R_i[p], R_j[q]) + 1$$
$$AL(p,q) = U(p,q)AL(p,q-1) + L(p,q)AL(p-1,q) + D(p,q)AL(p-1,q-1) + 1$$

where $m(R_i[p], R_j[q]) = 1$ if $R_i[p] = R_j[q]$ and $m(R_i[p], R_j[q]) = 0$ otherwise. Initial conditions are given by $ML(0,0) = AL(0.0) = 0, ML(0,q) = AL(0,q) = q, ML(p,0) = AL(p,0) = p$ for $p = 1, \ldots, l_i, q = 1, \ldots, l_j$. Figure 3 illustrates an example for the computation of the DP matrices M, U,D, L, ML and AL. The dark shaded cells and arrows show the optimal global alignment path. Note that this is a score-only computation and therefore requires only linear space.

Furthermore, we have employed the banded alignment concept to reduce the number of computed DP matrix cells. In this approach, only cells within a narrow band along the main diagonal are calculated. Using eight test datasets in Section 5 and a band of width 1/5 (1/10) of the length of the shorter read, pairwise genetic distances up to the threshold $\theta_g = 0.2$ (0.1) can still be computed accurately while the computation runtime is reduced by 2.29 (3.58) compared to the full alignment. However, read pairs with a genetic distance larger than 0.2 (0.1), might get a higher (but never lower) distance value assigned.

Even though some of the larger distance values might change, the pairwise distances can still result in an identical OTU structure after clustering. This is due to the fact that for most datasets used for microbial community profiling, we are only concerned about the species or genus level OTU assignment which assumes a distance threshold of 0.03 or 0.05 between reads within the same cluster. Though these distinctions are still controversial amongst microbiologists [14], the threshold of 0.2 can still ensure the correctness of the estimation.

An overview of the CUDA implementation on a single GPU of the genetic distance computation is shown in Figure 4. The pair indices and input reads are transferred to CUDA global memory, whereby reads are represented as binary strings using two bits per base: A=00, T=01, G=11, C=10.

Multiple CUDA threads can calculate the pairwise distances in parallel. During the computation, one row of DP matrix values per pairwise alignment is stored in CUDA global memory which is accessed using coalesced data transfer to reduce transfer time. Moreover, each thread within a thread block computes a DP matrix block of size 16×16 using fast access shared memory, which reduces the costly accesses to global memory by a factor of 16 and makes the kernel compute-bound rather than memory-bound. At the end of the computation, each thread returns a distance value to a buffer located in CUDA global memory. The result buffer is then transferred to host memory and the CPU creates the final sparse genetic distance matrix.

For the GPU cluster version, the sparse matrix of pair indices from the k-mer distance module is divided equally amongst the GPUs in the cluster through the host nodes. On each GPU, multiple CUDA thread blocks can perform the genetic distance computation in parallel since they are independent.

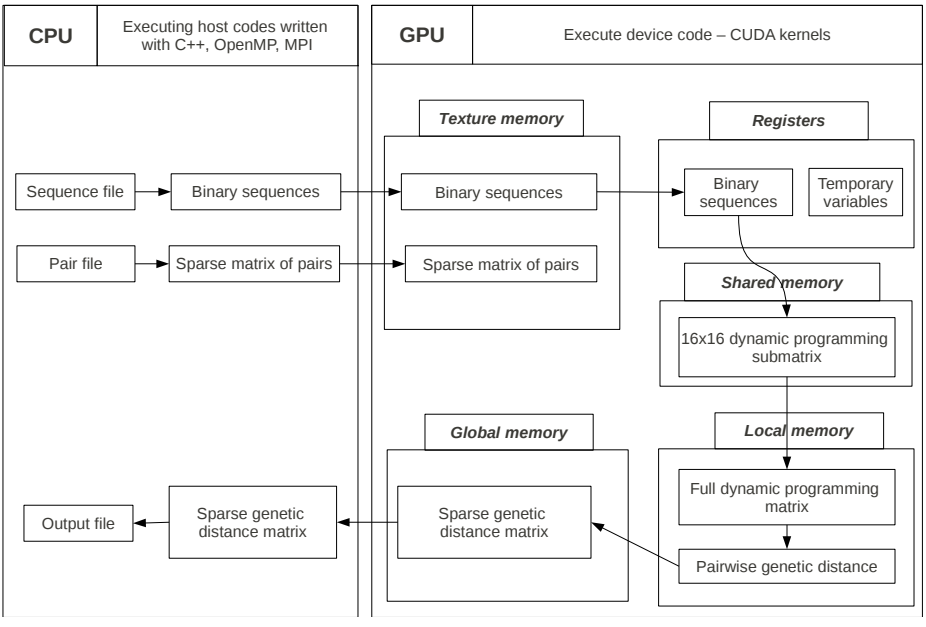

Fig. 4. CUDA implementation on GPU of the pairwise genetic distance computation

3.3 Space-Efficient Hierarchical Clustering

Hierarchical full linkage clustering is used for binning reads into OTUs from the linkage information provided by the sparse genetic distance matrix. Although the matrix is sparse, it is still of considerable size and often exceeds the amount of available RAM for large-scale datasets. Therefore, we have designed a memory-efficient hierarchical full linkage clustering implementation which can deal with sparse matrices of several hundred gigabytes in size. To reduce memory, ESPRIT [7] proposed the Hcluster algorithm using an "on-the-fly" strategy. Hcluster sorts the distances first and then shapes clusters by sequentially adding linkages. However, when the sparse matrix is very large the sorting procedure becomes a bottleneck. We have modified the Hcluster approach to make it more scalable.

Our approach first splits the sparse matrix into a number of smaller submatrices and then sorts each submatrix separately. We use a value queue to compensate for the fact that the entire sparse matrix is not fully sorted. Initially each submatrix file contributes one value (the smallest value) to the value queue. If the smallest value is removed from the queue by the clustering procedure, it is replaced by the next value from the corresponding file.

CRiSPy clustering is a procedure of shaping a binary tree, where each leave node presents a unique read. Pairwise distance values are scanned in ascending order. After a pairwise distance value is read from the value queue, the parent nodes of the two corresponding reads are located and the linkage information is added. Linkage information is only stored in parent nodes. This approach

is memory efficient and achieves a constant search time for the parent nodes regardless of the sparse matrix size. At the end of the computation, the OTUs with respect to the given genetic distance cutoff as well as the node linkage information are outputted. Unlinked reads and reads with first linkage at high distance level are outputted as outliers.

4 Results

To evaluate the performance of CRiSPy, we have acquired eight 16S rRNA pyrosequencing datasets from the NCBI Sequence Read Archive (SRA) including three medium datasets and five large datasets. These raw datasets usually contain reads of low quality which can introduce a considerable number of false diversity into the species richness estimation. Hence, we have preprocessed these raw datasets to remove reads which contains ambiguous nucleotides (N) and reads with lengths that are not within 1 standard deviation from the average length. The numbers of reads before and after preprocessing are recorded in Table 1.

We benchmark the performance of CRiSPy against ESPRIT, a software package for estimating microbial diversities using 16S rRNA pyrosequencing data written in C++. ESPRIT package includes two different versions: a personal computer version (ESPRIT PC) and a computer cluster version (ESPRIT CC). ESPRIT PC implements sequential codes running on a single-CPU. ESPRIT CC takes a data parallel approach to parallelize both k-mer and genetic distance computation. The input data is split into several parts and distributed to compute nodes in a cluster through job submission. The partial distance matrices are then combined to form the full sparse distance matrix which is clustered by the same clustering module as ESPRIT PC.

Table 1. Runtime (in secs) and speedup comparison of k-mer distance computation between ESPRIT and CRiSPy

Dataset	Number of raw reads	Number of cleaned reads	ESPRIT PC	ESPRIT CC		CRiSPy MT		Single-GPU		Quad-GPU	
			T	T	S	T	S	T	S	T	S
SRR029122	40864	15179	322	98	3.29	78	4.13	6.7	48	1.7	189
SRR013437	57902	22498	838	257	3.26	203	4.13	15.8	53	4.0	211
SRR064911	23078	16855	837	251	3.33	203	4.12	19.1	44	4.8	173
SRR027085	249953	131482	27600	8648	3.19	6014	4.59	473	58	119	232
SRR033879_81	1494071	300667	153000	48316	3.17	29130	5.25	3150	49	791	193
SRR026596_97	333630	178860	91300	29386	3.11	22432	4.07	1746	52	440	208
SRR058099	339344	162223	78000	25550	3.05	17920	4.35	1459	53	368	212
SRR043579	857248	256760	195000	66115	2.95	48221	4.04	4046	48	1015	192

Since the performance of ESPRIT CC depends on cluster setup and is subjected to communication overheads, we mainly use ESPRIT PC to benchmark the performance of CRiSPy. However, we also report the runtime and speedup of ESPRIT CC on a cluster of 4 CPUs to give readers a rough idea about the performance of ESPRIT CC. In this paper, we use the latest ESPRIT version released on February 2011. ESPRIT source code is available upon request to Dr. Sun Yijun (http://plaza.ufl.edu/sunyijun/ESPRIT.html).

We have implemented three different versions of CRiSPy:

1. a multithreaded C++ program with OpenMP (CRiSPy MT)
2. a CUDA program running on a single-GPU (CRiSPy single-GPU)
3. a CUDA program running on a multi-GPU cluster (CRiSPy multi-GPU)

We have conducted performance comparisons under the Linux OS with the following setup. ESPRIT PC and CRiSPy MT runtime are estimated on a Dell T3500 Workstation with a quad-core Intel Xeon 2.93 GHz processor, 4GB RAM. ESPRIT CC runtime is estimated on a cluster for four Dell T3500 Workstations connected via Ethernet. Condor High-Throughput Computing System is used for job submission. CRiSPy single-GPU runtime is measured on a Fermi-based Tesla S2050 GPU card with 3GB RAM. CRiSPy quad-GPU runtime is measured on a quad-GPU cluster which consists of two host nodes, each of which connected to two S2050 GPU cards. These nodes are connected by a high-speed Infiniband network. We use the following parameters for our experiments: $k = 6$, $\theta_k = 0.3$ for large datasets, $\theta_k = 0.5$ for medium datasets, $\theta_g = 0.2$, $sbt(x = y) = 5$, $sbt(x \neq y) = -4$, $\alpha = -10$ and $\beta = -5$.

4.1 Performance Comparison of *k*-mer Distance Computation

Table 1 records the runtime (in seconds) of the pairwise k-mer distance computation. For a dataset which contains n reads, the total number of pairs to be computed is $n(n-1)/2$. For both k-mer and genetic distance modules, T stands for runtime and S stands for speedup. T includes IO transfer time between host CPU and CPU, computation time of the algorithm and file IO time to output results to file.

CRiSPy MT is a multithreaded program written in C++ with OpenMP for a multi-core PC. It exploits the data parallelism of the pairwise distance matrix computation and achieves an average speedup 4.34 compared to ESPRIT PC. ESPRIT CC requires more time that CRiSPy MT mainly due to scheduling, communication overheads and data splitting stage. CRiSPy on a single-GPU (quad-GPU) runs 50.7(201.3) times faster than ESPRIT PC and 11.7 (46.4) times faster than CRiSPy MT.

4.2 Performance Comparison of Genetic Distance Computation

Table 2 shows the runtime (in minutes) of the genetic distance computation on the aforementioned datasets. The percentage of pairs p reported in Table 2 is defined as the number of read pairs with k-mer distance less than the threshold $\theta_k = 0.3$ divided by the total number of pairs which is $n(n-1)/2$ for a dataset of n reads. Hence,$1 - p$ is the percentage of pairs that k-mer distance module has effectively filtered out.

When running on multiple GPUs, the number of pair indices acquired from k-mer filtering step are divided equally amongst all the GPUs. Each GPU will then process its set of pair indices independently. As the runtime of ESPRIT PC, ESPRIT CC and CRiSPy MT are tremendous for large datasets, we sample representative parts of each dataset to get an estimated runtime.

Table 2. Runtime (in mins) and speedup comparison of genetic distance computation between ESPRIT and CRiSPy

Dataset	Average length	Percentage of pairs left p	ESPRIT PC	ESPRIT CC		CRiSPy MT		Single-GPU		Quad-GPU	
			T	T	S	T	S	T	S	T	S
SRR029122	239	11.04%	161	64	2.50	33	4.88	1.7	94	0.4	374
SRR013437	267	11.82%	462	184	2.51	94	4.93	4.6	101	1.1	402
SRR064911	524	56.73%	4700	1964	2.39	958	4.90	44	108	11	429
SRR027085	260	0.84%	1074	423	2.54	212	5.07	10	107	2.6	416
SRR033879_81	268	9.03%	64806	27029	2.40	12683	5.11	611	106	153	424
SRR026596_97	541	28.45%	2815	1128	2.50	559	5.04	26	110	6.5	434
SRR058099	531	6.76%	52430	22568	2.32	10551	4.97	479	109	120	437
SRR043579	556	8.37%	166160	73289	2.27	32627	5.09	1496	111	374	444

In comparison with ESPRIT PC, the runtime taken by CRiSPy MT reduces by the factor of 5.00 on average. Furthermore, average speedup gain by CRiSPy single-GPU (quad-GPU) is 105.6 (419.8) compared to ESPRIT PC and 21.1 (84.0) compared to CRiSPy MT. Similar to the filtration stage, ESPRIT CC encounters even more signification communication overheads since the amount of input and output data are much larger.

The speedup gain of the genetic distance module is much more significant compared to the k-mer distance module since genetic distance computation is more compute-intensive and hence it can utilize more processing powers of GPUs. Furthermore, the performance of this module increases in correspondence to the average length and the size of the input dataset.

4.3 Execution Time of CRiSPy Full Run

CRiSPy's processing pipeline includes three modules: parallel k-mer distance computation, parallel genetic distance matrix computation and sequential clustering of the resulting sparse distance matrix. Table 3 and Table 4 show the runtime of CRiSPy on a single-GPU and a quad-GPU cluster respectively for five large datasets.

For the genetic distance computation, we use three different global alignment schemes including full alignment, 1/5 banded alignment and 1/10 banded alignment. Please note that 1/5 (1/10) banded alignment reduces the runtime by a factor of 2.29 (3.58) compared to full alignment. We have observed from these experiments that 1/5 banded alignment produces identical OTUs as the full alignment and the 1/10 banded alignment results in only some minor difference in terms of outliers.

Table 5 shows the comparison between ESPRIT PC, CRiSPy MT and CRiSPy single-GPU full run for three medium datasets. We notice that by using banded alignment, CRiSPy single-GPU often requires two orders of magnitude less time than ESPRIT PC to execute the whole pipeline. Besides, the larger the dataset, the more significant speedup CRiSPy achieves.

Table 3. Runtime (in mins) of CRiSPy full run on a single-GPU

Dataset	k-mer distance computation	Genetic dist computation			Clustering		Total runtime		
		full	1/5	1/10	Sorting	Hcluster	full	1/5	1/10
SRR027085	7.9	10	4.8	3.3	2.6	3.0	24	18	17
SRR033879_81	53	611	286	195	162	285	1111	785	694
SRR026596_97	29	26	10	6.2	1.7	1.8	58	43	39
SRR058099	24	479	195	116	36	49	587	303	224
SRR043579	67	1496	607	360	102	152	1849	960	714

Table 4. Runtime (in mins) of CRiSPy full run on a quad-GPU cluster

Dataset	k-mer distance computation	Genetic dist computation			Clustering		Total runtime		
		full	1/5	1/10	Sorting	Hcluster	full	1/5	1/10
SRR027085	2.0	2.6	1.2	0.8	2.6	3.0	10	8.8	8.4
SRR033879_81	13	153	72	49	162	285	613	532	509
SRR026596_97	7.3	6.5	2.6	1.6	1.7	1.8	17	13	12
SRR058099	6.1	120	49	29	36	49	210	139	119
SRR043579	17	374	152	90	102	152	678	455	393

Table 5. Runtime (in mins) and speedup comparison between ESPRIT and CRiSPy

Dataset	ESPRIT PC	CRiSPy MT		Full band		1/5 band		1/10 band	
	T	T	S	T	S	T	S	T	S
SRR029122	167	35	4.79	2.4	68	1.6	106	1.3	125
SRR013437	477	98	4.85	6.1	78	3.7	130	3.0	157
SRR064911	4720	968	4.88	50	94	24	196	17	276

4.4 Assessment of Microbial Richness Estimation Accuracy by CRiSPy

To assess richness estimation accuracy and robustness of CRiSPy against sequencing errors, we have used several simulated datasets based on a dataset used by Huse et al. [15]. The simulated datasets have been generated by random sampling of reads from 43 known species and randomly introducing errors (insertion, deletion, mutation, ambiguous calls). Each of the simulated datasets contains ten thousand reads. A combination of error rates of 0.18% insertion, 0.13% deletion, 0.08% mutation and 0.1% ambiguous calls are used as standard error background. Besides 1/4, 1/2, 1 and 2 fold of the standard error rate are also introduced into the test dataset. Each test has been repeated ten times with newly simulated datasets and the average result is recorded.

Table 6 shows the estimated species richness for CRiSPy, ESPRIT, and MUSCLE+MOTHUR [5,16] for a varying amount of sequencing errors at a 0.03 and 0.05 genetic distance cutoff. The results show that MUSCLE+MOTHUR estimates a significantly higher richness at higher error rates than CRiSPy and ESPRIT. Furthermore, at the 0.03 genetic distance cutoff level, CRiSPy shows better stability than ESPRIT.

Table 6. Comparison of richness estimation of CRiSPy, ESPRIT and MUSCLE+MOTHUR for simulated datasets derived from 43 species

Tool	Genetic distance cutoff	Species estimated 1/4	1/2	1	2
CRiSPy	0.03	43	44	46	65
	0.05	42	42	43	43
ESPRIT	0.03	43	44	47	73
	0.05	42	42	43	43
MUSCLE+ MOTHUR	0.03	45	49	84	168
	0.05	44	44	54	80

5 Conclusion

In this paper, we present CRiSPy - a scalable tool for taxonomy-independent analysis of large-scale 16S rRNA pyrosequencing datasets running on low-cost hardware. Using a PC with a single CUDA-enabled GPU, CRiSPy can perform species richness estimation of input datasets containing over three hundred thousand reads in less than half a day. Based on algorithms which are designed for massively parallel CUDA-enabled GPUs, CRiSPy achieves speedup of up to two orders of magnitude over the state-of-the-art ESPRIT software for the time-consuming genetic distance computation step. Since large-scale microbial community profiling becomes more accessible to scientists, scalable yet accurate tools like CRiSPy are crucial for research in this area.

Although CRiSPy is designed for microbial studies targeting DNA sequence analysis, the individual k-mer distance and genetic distance modules on GPUs can easily be extended to support protein sequence analysis and be used in general sequence analysis studies such as the usage of k-mer distance for fast, approximate phylogenetic tree construction by Edgar [17] or the utilization of pairwise genetic distance matrix in multiple sequence alignment programs such as ClustalW [18].

Availability: CRiSPy is available from the authors upon request.

References

1. Sogin, M.L., Morrison, H.G., Huber, J.A., et al.: Microbial diversity in the deep sea and the underexplored rare biosphere. PNAS 103(32), 12115–12120 (2006)
2. Turnbaugh, P., Hamady, M., Yatsunenko, T., et al.: A core gut microbiome in obese and lean twins. Nature 457(7228), 480–484 (2009)
3. Fabrice, A., Didier, R.: Exploring microbial diversity using 16S rRNA high-throughput methods. Applied and Environmental Microbiology 2, 074–092 (2009)
4. Hamady, M., Knight, R.: Microbial community profiling for human microbiome projects: Tools, techniques, and challenges. Genome Research 19(7), 1141–1152 (2009)
5. Edgar, R.C.: MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research 32(5), 1792–1797 (2004)

6. Nawrocki, E.P., Kolbe, D.L., Eddy, S.R.: Infernal 1.0: inference of RNA alignments. Bioinformatics 25(10), 1335–1337 (2009)
7. Sun, Y., Cai, Y., Liu, L., et al.: ESPRIT: estimating species richness using large collections of 16S rRNA pyrosequences. Nucleic Acids Research 37(10), e76 (2009)
8. Edgar, R.C.: Search and clustering orders of magnitude faster than BLAST. Bioinformatics 26(19), 2460–2461 (2010)
9. Huse, S.M., Welch, D.M., Morrison, H.G., et al.: Ironing out the wrinkles in the rare biosphere through improved OTU clustering. Environmental Microbiology 12(7), 1889–1998 (2010)
10. Sun, Y., Cai, Y., Huse, S., et al.: A Large-scale Benchmark Study of Existing Algorithms for Taxonomy-Independent Microbial Community Analysis. Briefings in Bioinformatics (2011)
11. Liu, Y., Schmidt, B., Maskell, D.L.: CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. BMC Research Notes 3, 93 (2010)
12. Shi, H., Schmidt, B., Liu, W., et al.: A parallel algorithm for error correction in high-throughput short-read data on CUDA-enabled graphics hardware. Journal of Computational Biology 17(4), 603–615 (2010)
13. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48(3), 443–453 (1970)
14. Schloss, P.D., Handelsman, J.: Introducing DOTUR a Computer Program for Defining Operational Taxonomic Units and Estimating Species Richness. Applied and Environmental Microbiology 71(3), 1501–1506 (2005)
15. Huse, S.M., Huber, J.A., Morrison, H.G., et al.: Accuracy and quality of massively parallel DNA pyrosequencing. Genome Biology 8(7), R143 (2007)
16. Schloss, P.D., Westcott, S.L., Ryabin, T., et al.: Introducing MOTHUR Open-Source Platform-Independent Community-Supported Software for Describing and Comparing Microbial Communities. Applied and Environmental Microbiology 75(23), 7537–7541 (2009), doi:10.1128/AEM.01541-09
17. Edgar, R.C.: Local homology recognition and distance measures in linear time using compressed amino acid alphabets. Nucleic Acids Research 32(1), 380–385 (2004)
18. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Research 22(22), 4673–4680 (1994)