DDP-B: A Distributed Dynamic Parallel Framework for Meta-genomics Binary Similarity

Mengxian Chi(✉), Xu Jin(✉), Feng Li, and Hong An(✉)

University of Science and Technology of China, Hefei, China
{mxchi10,jinxu,fli186}@mail.ustc.edu.cn, han@ustc.edu.cn

Abstract. Great efforts have been made on meta-genomics in the field of new species exploration in the past decades. With the development of next-generation sequencing technology, meta-genomics datasets have been produced as large as dozens of hundreds of gigabytes or even several terabytes, which brings a severe challenge to data analysis. Besides, conventional meta-genomics comparing algorithms may not take full advantage of powerful computing capacity from parallel computing techniques due to lack of parallelism. In this paper, we propose DDP-B, a distributed dynamic parallel framework for meta-genomics binary similarity analysis, to overcome these limitations. In this framework, we introduce a binary distance algorithm for meta-genomics similarity measurement and develop different levels of parallel granularity of the algorithm utilizing MPI, OpenMP, and SIMD techniques. Moreover, we establish a dynamic scheduling method to deliver asynchronous parallel computing tasks and design a distributed cluster to deploy the dynamic parallel system, which completes 2.97K pairs of meta-genomics vectors comparison per second and achieves an 134.79x speedup versus the baseline in the optimal condition. Our framework shows stable scalability when assigned larger workloads.

Keywords: Meta-genomics · Big data · Parallel computing · Binary distance · Dynamic scheduling · Distributed scalability

1 Introduction

Great efforts have been made on exploring new species in the last several decades since the Woese significant work [23]. Meta-genomics [25], which involves the total DNA sequences extracted directly from the natural environment (e.g. ocean, soil, and the human body) samples, occasionally preserves the molecular signatures of potential unexplored or undiscovered microorganisms [24].

The work is supported by the National Key Research and Development Program of China (Grants No. 2016YFB1000403).

Published by Springer Nature Switzerland AG 2019
X. Tang et al. (Eds.): NPC 2019, LNCS 11783, pp. 143–155, 2019.
https://doi.org/10.1007/978-3-030-30709-7_12

Many novel techniques such as cultivation-independent shotgun genomics and next-generation sequencing [16] have been widely applied in this domain and dramatically aggrandized the scale of sequencing data as well as the speed of genome sequencing. As a result, meta-genomics datasets could be as large as dozens of hundreds of gigabytes or even several terabytes, which makes it a typical big data problem.

On the other hand, the rise of large scale computing clusters brings huge opportunities for meta-genomics research [2]. High-performance clusters support parallel computing and scalable architectures at multiple levels, which delivers extraordinary powerful performance theoretically. However, most of the conventional meta-genomics similarity algorithms barely take the most of high-performance parallel computing because there is a lack of excavation and utilization of their parallelism and scalability, which turns out the major limitations to deploy the algorithms on high-performance clusters. Moreover, the vast data scale brings a severe challenge to data storage and transmission in the field of high-performance computing.

Recently, binary distance measurements have been comprehensively applied in the field of biology [11], ethnology [6], and taxonomy [21]. Furthermore, genome sequence compressing methods (such as Hash map [18]) have contributed significantly to reducing the scale of meta-genomics datasets and enabling efficient search of massive sequences collections. Genome sequence data could be converted from character strings into binary vectors by the Hash map. As a result, we can measure the genome similarity through binary distance methods [5].

In this paper, we introduce a binary distance coefficient based comparing algorithm to measure meta-genomics similarity. The binary vectors generated from meta-genomics sequences is still too long (even more than 10^8 bits) to calculate the binary distance coefficient straightforward. To develop the algorithm efficiently, we divide a whole binary vector into 64-bit sub-sequences and process the calculation with Intel intrinsic instructions [15], which is easy to be paralleled as an atomic operation. Besides, we develop the binary similarity algorithm with hierarchical parallelism taking advantage of multiple parallel techniques such as SIMD [13], OpenMP [3], and MPI [10]. The hybrid parallel optimization delivers an 87.9x speedup compared with the original baseline.

Moreover, the data loading procedure is difficult to accelerate because of the memory read/write speed limitation, which constrains further optimization of the algorithm. And with the growth of data size, how to balance the workloads among large scale distributed clusters becomes a huge challenge [20]. To overcome these challenges, we design a dynamic scheduling system based on a master-slave structure and deploy it on a 9-node cluster. The scheduler distributes parallel computing tasks to the unoccupied worker nodes dynamically so that the communication and computation are decoupled and the data loading time is overlapped with computing time. As a result, we achieve a 7.5x speedup with 8 worker nodes under basic workload, which is close to linear acceleration. Meanwhile, we design a grouping strategy to organize worker nodes into extensive worker groups based on the workloads. Every worker group will be reorganized automatically if the workload exceeds its capacity. We achieve an

extra speedup benefit under 4 times of basic workload (15.55x versus 9.27x), which exhibits our framework having stable scalability under larger workloads.

With all the above contributions, we propose DDP-B, a distributed dynamic framework taking advantage of multiple parallel levels for a binary similarity algorithm. We deliver 2.97K pairs of meta-genomics similarity comparison per second and achieve an 134.8x speedup overall in the optimal condition using this framework.

The rest of the paper is organized as follows. Section 2 introduces related work as well as the background of our research. Section 3 explains our methodologies in detail. Section 4 presents the implementation and experimental results. Section 5 delivers a conclusion of whole work and discusses future research.

2 Background

In this section, we introduce some related techniques concerning genome comparing algorithms.

2.1 Genome Sequences Alignment

The next-generation sequencing techniques usually gather massively short genome reads and then align them into longer reads. Experimental evidence shows that a whole chromosome sequence usually covers millions to billions of base pairs [7,22]. Though it is difficult to compare the chromosomes from end to end because we can hardly find the exact beginning of them, there is a requirement to investigate the similarity between whole long genome reads. The main reason is that quite a few similar basic genome functional units may be carried by the chromosomes of many different organisms. So it is not easy to figure out whether the short reads belong to different species.

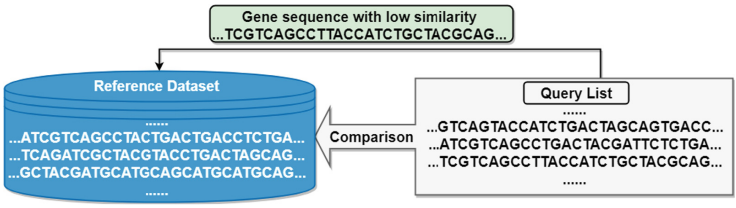

Fig. 1. Meta-genomics compares the new collected query genome list with the reference datasets, and extracts the genome sequences with low similarity to investigate whether there is a possibility of unknown species existing.

As meta-genomics sequences are captured from the environment randomly, it is almost impossible to extract every single microorganism's information through biological techniques. However, computer-aided analysis technology provides a

feasible method for the study of meta-genomics. Taking advantage of the redundant and overlapping information generated by genome sequencing techniques, genome fragments (also mentioned as *reads* by biologists) can be assembled into longer reads (also called *contigs*) and finally spliced into a chromosome sequence [14]. Consequently, meta-genomics research would be transformed into sequence alignment tasks [19]. We can speculate on the possibility of the existence of unknown organisms based on the results of sequence alignment. Potential undiscovered species information will be dug out through genome sequence compare processing if there exists a quite different sequence compared with every known reference sequence. Figure 1 shows the meta-genomics comparing processing.

2.2 K-mer, Hash Map, and Binary Distance

The conventional comparing algorithms designed to quantitatively evaluate the similarity among meta-genomics based on computing system usually regard genome sequences as character strings (i.e. representing DNA's four bases with 'A', 'G', 'C', and 'T' and assembling them into strings in order.) and compare these strings to measure the similarities among genomes. Therefore, numerous algorithms of string similarity comparison have been applied on meta-genomics, which can be roughly divided into two categories: exact matching (such as Boyer Moore Algorithm and Shift Or Algorithm [4]) and approximate matching (such as Edit Distance and Haiming Distance [17]).

Except for the naive sequence comparing algorithm, there are also many other distinguished methods obtaining remarkable achievements [4,17]. Among them, K-mer similarity [1] is widely applied in bioinformatics which generates k-length sub-sequences of a long read step by step, therefore, we could just compare the much shorter K-mers. Although an L-length read still produces $L-k+1$ K-mers, we could focus on the distinct K-mers, so that the scale of the datasets will be reduced appreciably.

Besides, K-mers of a whole genome sequence can be mapped into a binary vector using Hash map algorithms. Figure 2 shows the detail of K-mers Hashing conversion. The vector's i-th position will be set to 1 only if the Hash value of the K-mer equals to i, which is expressed as

$$Vec[i] = \begin{cases} 1, & Hash(K\text{-}mer) = i \\ 0, & others \end{cases} \tag{1}$$

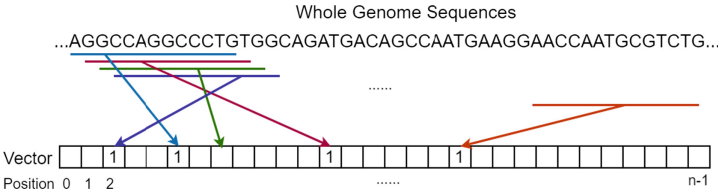

Fig. 2. Convert K-mers into binary vector. The vector's i-th position will be set to 1 only if the Hash value of the K-mer equals to i.

Therefore, the meta-genomics similarity problem is transformed into a binary distance problem. Choosing the appropriate Hash function is not within the scope of this paper. We focus the research on how to take advantage of the parallelism and the scalability of the binary distance algorithm.

3 Methodologies

3.1 Binary Distance Coefficient

The binary distance method can be regarded as an approximate matching algorithm and applied on the domain of meta-genomics with two major advantages: (1) the inaccuracy results are almost surely generated during the genome sequencing procedure limited by the transcription properties of genetic information, so that the approximate methods can provide sufficient effectiveness; (2) data size and computational complexity are likely to be obviously reduced so that the approximate algorithms usually perform more efficiently. In the field of binary distance research, the Jaccard coefficient [12] is one of the most famous measurements and the Forbes coefficient [8] is proposed for clustering ecologically related species especially, so that the Forbes coefficient could reveal the genome similarity quantitatively. In this paper, we define a modified Forbes-II coefficient to measure the similarity score between the genome binary vectors.

$$S_{M-F_{II}} = \frac{na - (a+b)(a+c)}{(a+b)^2 + (a+c)^2 - (a+b)(a+c)} \tag{2}$$

where the definitions of n, a, b, c, and d are referred to Table 1. The coefficient $S_{M-F_{II}}$ is only related to the parameters $n, a, (a+b), (a+c)$. In other words, the binary distance between two genome vectors depends on the length of two vectors, the number of bit set to 1 (abbreviated as bit-1) inside the bit-wise logic AND result from two vectors, and the number of bit-1 inside each vector respectively.

Table 1. Vec 1, 2 are two n-length binary vectors, a is the number of attributes where the values of Vec 1 and Vec 2 are both 1, b is the number of attributes where the value of Vec 1 and Vec 2 is (0,1), c is the number of attributes where the value of Vec 1 and Vec 2 is (1,0), and d is the number of attributes where both Vec 1 and Vec 2 have 0.

&		Vec 1		
		1	0	Sum
Vec 2	1	a	b	a + b
	0	c	d	c + d
	Sum	a + c	b + d	n = a + b + c + d

Algorithm 1. Binary Coefficient Calculating Parallel Hierarchy

Phase 0:
Receive $Query_{[p_i:p_j]}[0:n]$, Allocate $QueryBit_{[p_i:p_j]} \leftarrow \{0\}$
#pragma omp parallel
for m in $[p_i : p_j]$ **do**
  #pragma simd
  $QueryBit_{[m]}$ += $popcnt(Query_{[m]}[0:n])$
end for
Phase 1: Send Ready Signal
Phase 2:
Receive $RefVec_{[q_i]}[0:n]$, Allocate $RefBit_{[q_i]} \leftarrow 0$, $AndBit_{[q_i]}[p_i : p_j] \leftarrow \{0\}$
Allocate $S_{M-F_{II}[q_i][p_i:p_j]} \leftarrow \{0\}$
#pragma simd
$RefBit_{[q_i]}$ += $popcnt(RefVec_{[q_i]}[0:n])$
#pragma omp parallel
for m in $[p_i : p_j]$ **do**
  #pragma simd
  $AndBit_{[q_i]}[m]$ += $popcnt(Query_{[m]}[0:n]$ & $RefVec_{[q_i]}[0:n])$
end for
Calculate $S_{M-F_{II}[q_i][p_i:p_j]}$
Phase 3: Send $S_{M-F_{II}[q_i][p_i:p_j]}$, Ready Signal

3.2 Parallel Hierarchy Design

Therefore, how to count the number of bit-1 inside a binary vector becomes the first challenge. Here we evaluate three counting methods: *left shift*, *look-up table*, and *popcnt*. The left shift method shifts the vector to the left continuously and counts the number of bit-1 according to the sign bit. The look-up table means preparing a table consisting of the number of bit-1 in advance and looking for the exact number based on the binary vector's decimal form. Both of them are inefficient because they are both at a computation complexity of $O(n)$ for an n-bit vector. Besides, the look-up table method requires an extra $O(2^n)$ memory consumption.

Intrinsic Instruction Optimization. Fortunately, Intel intrinsic instructions provides an operator to count the number of bit-1 inside a 64-bit unsigned integer (named *popcnt*) bound with an assembly instruction. So that the theoretical computation complexity of this operation is $O(1)$ for a 64-bit vector. We have measured all the above methods and find that the performance of the popcnt is at least 2x faster than the other two methods. Moreover, popcnt is easy to parallelize as an atomic operation. Therefore, we use the popcnt operator to calculate the number of bit-1 inside vectors.

Data-Level Parallel. Since the length of binary genome vectors usually exceeds the capacity of the popcnt operator (64-bit), we need to separate a whole vector into several 64-bit sub-vectors. Here we utilize the SIMD (Single Instruction Multiple Data) vectoring techniques to deal with two 64-bit binary sub-vectors

simultaneously as the Intel AVX (Advanced Vector Extensions) supplies 128-bit registers. This method could deliver a 2x speedup theoretically.

Thread-Level Parallel. When comparing long-winded binary vectors, we allocate the popcnt and bit-wise AND operations to multiple threads uniformly as the two operations are both regular and aligned. Here we apply compiler directives of OpenMP (Open Multi-processing) to provide multi-threading parallel with a portable, scalable model. Meanwhile, we fork and synchronize the threads dynamically to balance the workloads among threads.

Overall, we accumulate each parameter and organize the binary distance coefficient computation through different levels of parallel granularity. Algorithm 1 shows the detail of the parallel hierarchy to calculate the distance coefficient.

3.3 Distributed Dynamic Schedule Design

We construct the dynamic scheduling system based on two major components in this system: Master Node and Slave Nodes. The communication interface between the master node and the slave nodes is established by MPI (Message Passing Interface). Figure 3 shows the distributed scalable dynamic programming architecture and exhibits the detail of control flow. And we will describe the behavior of the master node detailedly in the following content.

Master Node. It is the core module of our distributed dynamic system. The master node is responsible for managing all worker groups, fetching binary vectors from reference and query lists, broadcasting vectors to targeted worker

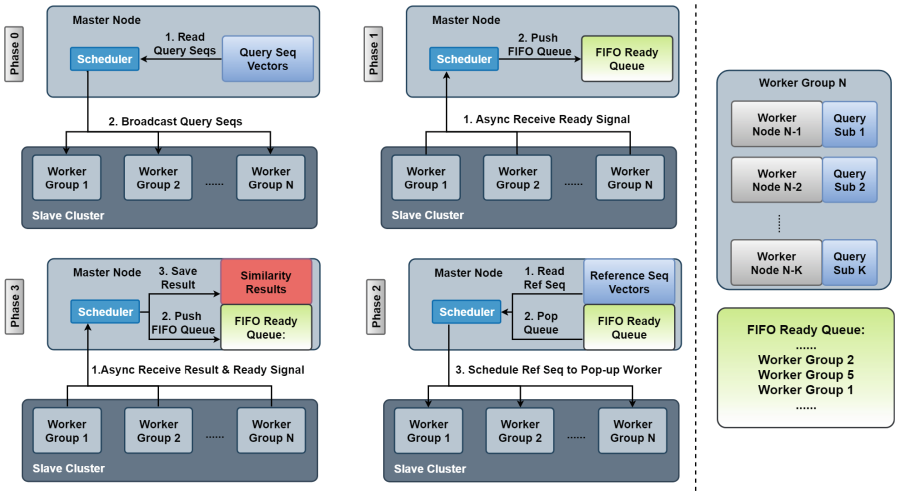

Fig. 3. Distributed scalable dynamic programming. The left part shows the data flow path of dynamic scheduling. The right part shows the worker group structure and FIFO ready queue.

groups, and gather all comparing results from them. We design two kernels for the master node: *Ready Queue* and *Scheduler*.

Ready Queue is a two-way first-in-first-out (FIFO) queue, preserving the standby state information of worker groups. A new ready worker group will be pushed back to the end of the ready queue, while an assigning worker group will be popped up from the top.

Scheduler is the controller unit and decides all behaviors of the master node. The control flow can be divided into four asynchronous phases as below. The scheduler will scan through these phases until all the comparing tasks distributed and all calculating results gathered.

- Phase 0: Scheduler fetches the query vectors while the query list is not empty and then broadcasts them to all worker groups. Every worker group will keep a complete copy of the query list.
- Phase 1: Every worker group sends a ready signal to scheduler asynchronously after receiving all the query vectors. Scheduler pushes the worker group ID into the end of the FIFO ready queue once receives the response signal from the worker.
- Phase 2: Scheduler pops a worker group ID from the ready queue and distributes one reference vector to the popped worker group while the reference list is not empty.
- Phase 3: After finishing the comparing task, the worker group sends the similarity results as well as a ready signal back to the scheduler. The scheduler will save the results with a tag and push the ready worker into the queue again.

Slave Nodes. This is the workload module which undertakes all the calculating tasks. We construct slave nodes as a set of scalable *worker groups*. Slave nodes have a flexible structure and can be recombined according to the real workload in practice.

Worker Group. It is assembled with one or more worker nodes to keep a complete copy of the query list with all memory consumption. And we separate the whole query list into k subsets if there are k worker nodes in one worker group. Every worker group will be reorganized automatically if the scale of query data exceeds this group's capacity.

Worker Node. We make each worker node keep a subset of the query list in its local memory. At Phase 3, every worker node compares one piece of reference vector with all the query vectors in the subset to calculate the similarity coefficient following Algorithm 1.

Grouping Strategy. Distributed heterogeneous clusters grouping strategy could be regarded as a typical knapsack problem (which is an NP-complete problem), so we will leave it for further research. For simplicity of implementation, in this paper we adopt a *naive grouping strategy* to organize worker nodes, i.e. we will add worker node one by one from scratch into a worker group until the worker group's capacity is enough to hold a whole copy of query list.

4 Experiment Results

4.1 Implementation

In order to deploy our framework, we adopt a 9-nodes cluster to build a dynamic system with 1 master node and 8 worker nodes. There are two CPU sockets on each node, 8 physical cores with hyper-threading enabled in each CPU. Every worker group has one worker node in default. The master node is connected with every worker node by Infiniband. Other hardware and software information is provided in Table 2.

Table 2. Hardware and software information

Item	Description
CPU	Intel(R) Xeon(R) E5-2660 @ 2.2 GHz * 2
Memory	DDR3 1333 MHz 96 GB
Hard disk	SAS HDD 300 GB
Network	Intel Ethernet Adapter I350 with 1 Gb/s
Connection	Mellanox QDR Infiniband 40 Gb/s
Operating system	CentOS 7.2-1511 Linux 3.10.0
MPI	Intel MPI 2017.0.098

The binary vectors are aligned at 3×10^8-bit length, which is supplied by DOE Joint Genome Institute [9]. As mentioned in Sect. 3.2, we separate each vector into 4.69M 64-bit sub-vectors. So one $S_{M-F_{II}}$ coefficient computation requires about 4.69M bit-wise ANDs, 14.06M popcnts, and 18.75M accumulations (37.5M operations altogether). We set both query list and reference list to 203 vectors so that the traverse comparison requires 1.55T operations. Every experiment is repeated three times and adopted the average results to avoid environmental instability. The serial computing baseline completes 22.07 coefficients calculating per second.

4.2 Performance Analysis

We summarize the runtime performance of the distributed dynamic programming system in a parallel hierarchy.

Multi Threads. Figure 4 shows the multi-threading performance optimized on a single node. We observe that the speedup curve has a sub-linear growth trend when threads number goes from one to eight. And then, it will convergence and even suffer performance damage after that. As mentioned above, every CPU has eight physical cores. Hence CPU may be overloaded when forked into more than eight threads which increases the overhead of thread switching. The multithreading delivers 4.2x speedup with 16 threads compared with baseline.

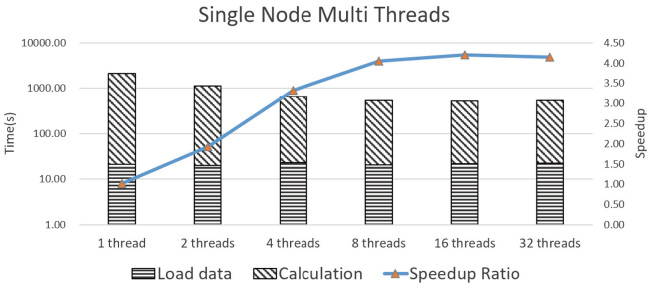

Fig. 4. Single Node Performance. The bar chart uses exponential coordinates for clarity.

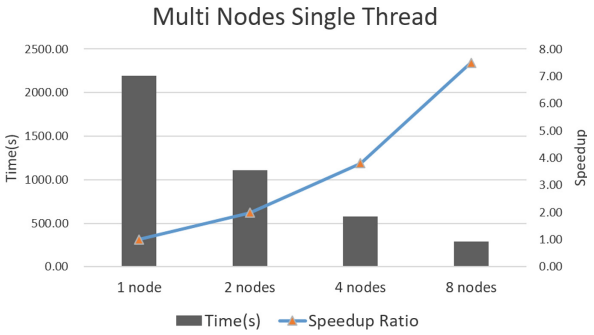

Fig. 5. Multi Nodes Single Thread Performance. The master node is not counted.

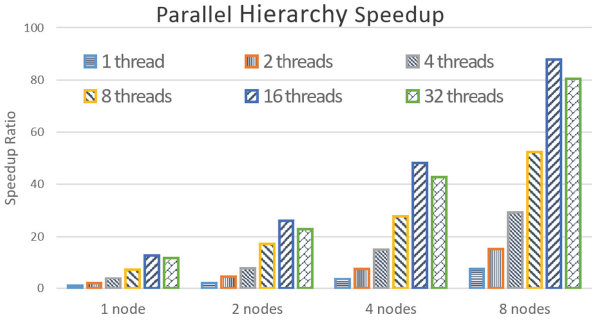

Fig. 6. Hybrid Parallel Performance. The bar chart shows the speedup ratio compared with single node single thread baseline.

Multi Nodes. Moreover, we evaluate the data loading time and coefficient computing time consumption separately. Although the data loading procedure is not time-consuming, it is difficult to parallelize this part because of the memory read/write speed limitation. However, we can efficiently overlap this procedure with computing in multi-nodes architecture. Figure 5 shows the benefits from multi-nodes single thread parallel. The speedup curve demonstrates a favorable

accelerating trend and this method delivers 7.5x speedup compared with baseline as a result.

Hybrid Parallel. Then we nest MPI and OpenMP programming techniques to take the most of this distributed dynamic architecture. Figure 6 exhibits the comprehensive results with a various number of nodes as well as threads. The optimized performance trends similarly with those mentioned above two orthogonal strategies. System performance increases with the number of nodes as well as threads but will degenerate when too many threads assigned on one worker node. The optimal performance is achieved on 8 nodes 16 threads, which gives an 87.9x speedup compared with the baseline.

Table 3. SIMD performance

8 nodes 16 threads	With/SIMD	With SIMD	Speedup
Time	21.25 s	13.85 s	1.5x

Table 4. Scalability under larger workloads

Query vecs	Reference vecs	1 node 16 threads with SIMD	8 nodes 16 threads with SIMD	Speedup
203	203	128.42 s	13.85 s	9.27x
203	812	506.54 s	32.58 s	15.55x

SIMD. We investigate the SIMD optimization as well as the scalability of our system. Table 3 shows that an extra 1.5x speedup obtained by the SIMD on multi-nodes multi threads condition approaching the theoretical 2x peek.

Scalability. Furthermore, we enlarge the reference data size to examine the scalability of the system. As Table 4 shows, our system achieves a 15.55x speedup under a larger workload while the original speedup is 9.27x when we keep 16 threads on every worker node and scale the number of worker nodes from 1 to 8 which exhibits super-linear scalability.

In a summary, we accomplish 2.97K binary coefficients calculating per second which gives an 134.8x speedup compared with baseline as well as 111.6 GOPS (Giga Operations per Second) at the condition of 8 worker nodes, 16 threads per worker node with SIMD applied.

5 Conclusion

In the filed of computer-aid meta-genomics research, how to design similarity measurement algorithms with high efficiency remains an enormous challenge.

In this paper, we propose PPD-B, a distributed dynamic parallel framework based on a binary similarity coefficient to support the meta-genomics analysis. Our framework modifies the Forbes coefficient to quantitatively evaluate the similarity among Hashed meta-genomics binary vectors and utilizes a hierarchical parallel architecture to optimize the computing process of coefficients computation. The experimental results show that the framework operates efficiently and achieves an 134.8x speedup compared with the baseline. And we design a scalable distributed dynamic programming system scheduling the whole system to decouple the communication and computation, which proven stable scalability on large workloads. Our work can be a novel standard instance implicating for designing efficient meta-genomics algorithms on distributed parallel clusters.

In the future, we plan to further research on several respects: assembling heterogeneous machines into our dynamic architecture with balanced workloads, which can be regarded as a typical knapsack problem; applying our system on multi-core accelerators or processors such as GPUs and Sunway to accelerate our algorithm; investigating more efficient binary similarity measurements and related Hash algorithms; and comparing our framework with prospective meta-genomics similarity analysis systems in terms of performance and effectiveness.

References

1. Bernard, G., Greenfield, P., Ragan, M.A., Chan, C.X.: k-mer similarity, networks of microbial genomes, and taxonomic rank. mSystems **3**(6), e00257–18 (2018)
2. Buyya, R., et al.: High Performance Cluster Computing: Architectures and Systems (Volume 1), vol. 1, p. 999. Prentice Hall, Upper Saddle River (1999)
3. Chapman, B., Jost, G., Van Der Pas, R.: Using OpenMP: Portable Shared Memory Parallel Programming, vol. 10. MIT Press, Cambridge (2008)
4. Charras, C., Lecroq, T.: Handbook of Exact String Matching Algorithms. Citeseer (2004)
5. Choi, S.S., Cha, S.H., Tappert, C.C.: A survey of binary similarity and distance measures. J. Syst. Cybern. Inform. **8**(1), 43–48 (2010)
6. Driver, H.E., Kroeber, A.L.: Quantitative Expression of Cultural Relationships, vol. 31. University of California Press, Berkeley (1932)
7. Fleischmann, R.D., et al.: Whole-genome random sequencing and assembly of haemophilus influenzae RD. Science **269**(5223), 496–512 (1995)
8. Forbes, S.A.: On the local distribution of certain Illinois fishes: an essay in statistical ecology, vol. 7. Illinois State Laboratory of Natural History (1907)
9. Grigoriev, I.V., et al.: The genome portal of the department of energy joint genome institute. Nucleic Acids Res. **40**(D1), D26–D32 (2011)
10. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Comput. **22**(6), 789–828 (1996)
11. Hubalek, Z.: Coefficients of association and similarity, based on binary (presence-absence) data: an evaluation. Biol. Rev. **57**(4), 669–689 (1982)
12. Jaccard, P.: Étude comparative de la distribution florale dans une portion des alpes et des jura. Bull. Soc. Vaudoise Sci. Nat. **37**, 547–579 (1901)
13. Jeong, H., Kim, S., Lee, W., Myung, S.H.: Performance of SSE and AVX instruction sets. arXiv preprint arXiv:1211.0820 (2012)

14. Li, D., Liu, C.M., Luo, R., Sadakane, K., Lam, T.W.: Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. Bioinformatics **31**(10), 1674–1676 (2015)
15. Lomont, C.: Introduction to Intel advanced vector extensions. Intel White Paper, pp. 1–21 (2011)
16. Metzker, M.L.: Sequencing technologies-the next generation. Nat. Rev. Genet. **11**(1), 31 (2010)
17. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. (CSUR) **33**(1), 31–88 (2001)
18. Ondov, B.D., et al.: Mash: fast genome and metagenome distance estimation using minhash. Genome Biol. **17**(1), 132 (2016)
19. Rognes, T., Flouri, T., Nichols, B., Quince, C., Mahé, F.: Vsearch: a versatile open source tool for metagenomics. PeerJ **4**, e2584 (2016)
20. Schroeder, B., Gibson, G.: A large-scale study of failures in high-performance computing systems. IEEE Trans. Dependable Secur. Comput. **7**(4), 337–350 (2009)
21. Sneath, P.H.A.: The principles and practice of numerical classification. Numer. Taxon. **573**, 263–268 (1973)
22. Wilming, L.G., Gilbert, J.G., Howe, K., Trevanion, S., Hubbard, T., Harrow, J.L.: The vertebrate genome annotation (vega) database. Nucleic Acids Res. **36**(suppl_1), D753–D760 (2007)
23. Woese, C.R., Fox, G.E.: Phylogenetic structure of the prokaryotic domain: the primary kingdoms. Proc. Natl. Acad. Sci. **74**(11), 5088–5090 (1977)
24. Woyke, T., Rubin, E.M.: Searching for new branches on the tree of life. Science **346**(6210), 698–699 (2014)
25. Wrighton, K.C., et al.: Fermentation, hydrogen, and sulfur metabolism in multiple uncultivated bacterial phyla. Science **337**(6102), 1661–1665 (2012)