Computing the Diameter of 17-pancake Graph using a PC Cluster

Shogo Asai[1], Yuusuke Kounoike[1], Yuji Shinano[1], and Keiichi Kaneko[1]

Tokyo University of Agriculture and Technology, Tokyo 184-8588, Japan,
asai@al.cs.tuat.ac.jp,{kounoike,yshinano,k1kaneko}@cc.tuat.ac.jp.
WWW home page: http://opt.cs.tuat.ac.jp/

Abstract. An n-pancake graph is a graph whose vertices are the permutations of n symbols and each pair of vertices are connected with an edge if and only if the corresponding permutations can be transitive by a prefix reversal. Since the n-pancake graph has $n!$ vertices, it is known to be a hard problem to compute its diameter by using an algorithm with the polynomial order of the number of vertices. Fundamental approaches of the diameter computation have been proposed. However, the computation of the diameter of 15-pancake graph has been the limit in practice. In order to compute the diameters of the larger pancake graphs, it is indispensable to establish a sustainable parallel system with enough scalability. Therefore, in this study, we have proposed an improved algorithm to compute the diameter and have developed a sustainable parallel system with the Condor/MW framework, and computed the diameters of 16- and 17-pancake graphs by using PC clusters.

1 Introduction

In this paper, let us consider a problem in which a stack of pancakes whose sizes are completely different is rearranged so that the pancakes form a pile where the sizes of pancakes increase from the top to the bottom. As operations of rearrangement, reversing several pancakes from the top of the stack is possible. The problem to obtain the largest number of operations to rearrange the worst-case stack of n pancakes as a function of n is called the pancake sorting problem[1]. This problem is also called the prefix reversal problem.

A pancake graph is a graph whose vertices are the permutations of n symbols from 1 to n and its edges are given between permutations transitive by prefix reversals. Since the graph topology is dependent on n, it is called an n-pancake graph. An n-pancake graph is a regular graph that has $n!$ vertices and its degree is $n-1$. The pancake sorting problem and the problem to obtain the diameter of the pancake graph is equivalent. Since the pancake graphs have many merits such as the symmetric and recursive structures, and the small degrees and diameters against the sizes, much attention is paid to them as a model of interconnection networks for parallel computers[2–4]. When we regard the pancake graphs as the model of the interconnection networks, the diameter of the graph is a measure that represents the delay of communication[5, 6].

Table 1. The diameters of n-pancake graphs

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Diameters	0	1	3	4	5	7	8	9	10	11	13	14	15	16	17

To obtain the diameter of an n-pancake graph, it is sufficient to obtain the shortest distances from one vertex to all the vertices. However, the algorithms that depend on the numbers of vertices and/or edges cannot solve the problem practically because the computational time and the memory space increase exponentially. Hence, Kounoike et *al.*[7] proposed a method that restricts the number of vertices for which the shortest distances must be obtained by taking advantage of the recursive structure of the pancake graphs. This method is based on the method by Heydari et *al.*[8] to obtain the diameter of the 13-pancake graph and is extended not to execute the unnecessary search. Kounoike has applied the method to give the diameters of 14- and 15-pancake graphs that were unknown so far. Table 1 shows the known diameters of the pancake graphs. Some attentions are paid to the sequence of diameters mathematically, and the sequence up to $n = 13$ is listed in the 'On-Line Encyclopedia of Integer Sequences'[9] as 'Sorting by prefix reversal.' However, no sequence for $n \geq 14$ is listed there. Hence, obtaining the diameters of the larger pancake graphs also contributes the study of the sequences.

In this study, we have improved the method by Kounoike et *al.* when they obtained the diameter of 15-pancake graph so that it computes the diameters of the larger pancake graphs and implemented it as a parallel computing system. In addition, we made use of the implemented system to obtain the diameters of 16- and 17-pancake graphs that have been unknown.

2 Definitions of Terminology and Symbols

In this section, we define the terminology and symbols used in this paper. Refer [7] for the detailed explanations.

Let S_n be the set of all the permutations of n symbols from 1 to n, and let the symbols 1 to n correspond to the smallest size of pancake to the largest one. Then assume that a permutation $\pi \in S_n$ which is obtained by arranging the symbols from the top pancake to the bottom pancake represents a stack of n pancakes. Let $\boldsymbol{e}_n$ be the permutation $(1, 2, \ldots, n)$ that corresponds to the sorted stack. Let $\sigma \in S_n$ be a permutation that is obtained by reversing the preceding k $(2 \leq k \leq n)$ symbols in $\pi \in S_n$. Then the transformation from the permutation π to the permutation σ is called the prefix reversal of k symbols for the permutation π, and it is denoted $\pi^k = \sigma$. Since we use only the prefix reversals of permutations in this paper, we mention reversals to mean the prefix ones. The successive reversals $(\pi^{x_1})^{x_2}$ of a permutation π are also denoted $\pi^{(x_1,x_2)}$. Moreover, if $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ then let $\pi^{\boldsymbol{x}}$ represent a successive reversals with x_1, $x_2, \ldots, x_m$ symbols. If $\pi^{\boldsymbol{x}} = \boldsymbol{e}_n$ then $\boldsymbol{x}$ is called a sorting sequence of π. For a given permutation $\pi \in S_n$, let the function $f(\pi) = \min\{|\boldsymbol{x}| : \pi^{\boldsymbol{x}} = \boldsymbol{e}_n\}$ represent the smallest number of reversals to sort the permutation. In addition, let the function $f(n) = \max\{f(\pi) : \pi \in S_n\}$ represent the largest number of

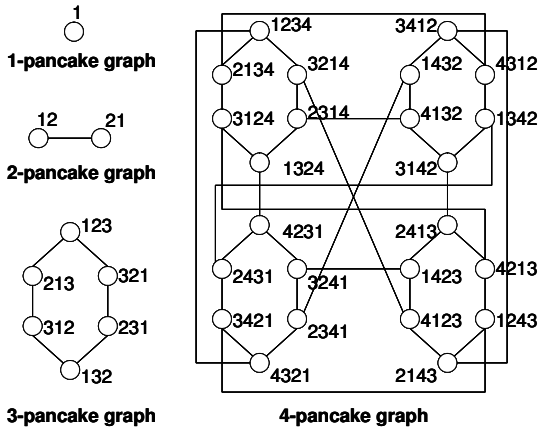

Fig. 1. The pancake graphs

reversals to sort the stacks of n pancakes. Conventionally, the same letter f is used for the functions. Note that the meaning of f depends on its argument.

A pancake graph is a graph whose vertices are $\pi \in S_n$, and whose edges are between vertices π and σ where $\sigma = \pi^k$. Since pancake graphs are different depending on n, each pancake graph is called n-pancake graph and denoted by P_n. Figure 1 shows P_1 to P_4. In general, between two vertices in a graph, the path that has the smallest number of edges is called the shortest path between the two vertices, and the number of edges included in the path is called the shortest distance. For arbitrary pair of two vertices in a graph, the longest shortest distance is called the diameter of the graph. By selecting $\boldsymbol{e}_n$ as one of the pair of vertices to which we compute the shortest distance, computing the diameter of an n-pancake graph is equivalent to computing $f(n)$. In this paper, the shortest distance between a vertex $\pi \in S_n$ and the vertex $\boldsymbol{e}_n$ is simply mentioned the distance of π.

3 Basic Method

We took the method by Kounoike et *al.*[7] by which they obtained $f(15)$ as the basic method to obtain the diameters. The method obtains the dependency between vertices based on the symmetric and recursive properties of pancake graphs and restricts the vertices whose distance computation is necessary.

First, for a permutation $\pi = \boldsymbol{e}_{n-1}^{\boldsymbol{x}} \in S_{n-1}$, we define a permutation $\sigma_k \in S_n$ $(1 \leq k \leq n)$ by expression (1). Then, for $f(\sigma_k)$, expression (2) holds.

$$\sigma_k = \begin{cases} ((\boldsymbol{e}_n)^n)^{\boldsymbol{x}} & k = 1 \\ ((\boldsymbol{e}_n)^{(k,n)})^{\boldsymbol{x}} & 2 \leq k \leq n-1 \\ \boldsymbol{e}_n^{\boldsymbol{x}} & k = n \end{cases} \tag{1}$$

$$f(\sigma_k) \leq \begin{cases} f(\pi)+1 & k = 1 \\ f(\pi)+2 & 2 \leq k \leq n-1 \\ f(\pi) & k = n \end{cases} \tag{2}$$

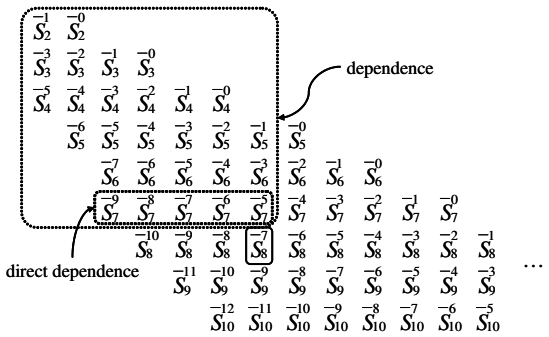

Fig. 2. The dependency relation between $\overline{S}_n^m$

For $\pi \in S_{n-1}$, let $T_k(\pi)$ be the transformation that obtains $\sigma_k \in S_n$, and let $T_k(S)$ be a set of $T_k(\pi)$ for all the elements of the set $S \subseteq S_{n-1}$. In addition, let S_n^m be a set of $\pi \in S_n$ such that $f(\pi) = m$ holds where S_n^k be empty for k such that $k < 0$ or $k > f(n)$. Then define the set $\overline{S}_n^m$ by expression (3).

$$\overline{S}_n^m = T_1(S_{n-1}^{m-1}) \cup T_2(S_{n-1}^{m-2}) \cup \cdots \cup T_{n-1}(S_{n-1}^{m-2}) \cup T_n(S_{n-1}^m) \tag{3}$$

This is the set of vertices in S_n whose upper bounds are equal to m. Then, from expression (2), $f(\pi) \leq m$ holds for $\pi \in \overline{S}_n^m$. The following relation holds among $\overline{S}_n^m$, S_n^m and S_n:

$$S_n = \bigcup_{k=0}^{f(n-1)+2} \overline{S}_n^k, \tag{4}$$

$$S_n^m \subseteq \bigcup_{k=m}^{f(n-1)+2} \overline{S}_n^k. \tag{5}$$

From expression (4), we can see that

$$f(n) \leq f(n-1) + 2. \tag{6}$$

In Figure 2, a set depends on the sets that are just above or upper left of it, and the lower left and upper right blank parts represent empty sets. We cannot judge if the below part of a set is empty or not until its diameter is computed. Based on the relationship, we can obtain an arbitrary $\overline{S}_n^m$ by repeating transformation and distance computation recursively from $S_1 = S_1^0 = \{\boldsymbol{e}_1\}$. To obtain the diameter $f(n)$, we first obtain $f(\pi)$ for all of $\pi \in \overline{S}_n^{f(n-1)+2}$. Then, depending on the existence of π that satisfies $f(\pi) = f(n-1) + 2$, $f(n)$ is classified as follows:

- In case that π which satisfies $f(\pi) = f(n-1)+2$ exists: $f(n) = f(n-1)+2$ holds. We can finish computation just after such π is found (See expression (6)).
- Otherwise: $f(n) \leq f(n-1) + 1$ holds. We can finish computation with the result $f(\pi) = f(n-1) + 1$ by showing π which satisfies the equation.

For searching shortest paths, we use A* algorithm. Refer [7] to see the detail of the algorithm.

The implementation by Kounoike et *al.* fixes the elements of the sets in Figure 2 from the leftmost column by performing transformation and distance computation. The diameters are also obtained in the process. This search method makes it possible to skip the searches of vertices that are known to be unnecessary for diameter computation based on dependency among the sets. However, as the size of the pancake graph increases, the number of elements in the sets becomes very large, and we cannot manage the pancake graph only with the main memory. Their implementation stores all the results of distance computation for later use. The results increase exponentially, and it occupies 21GB of the disk as a compressed file after the computation of $f(14)$. Hence, their implementation has the limitation for diameter computation of the larger pancake graphs.

4 Our New Implementation

In the previous implementation, it is impossible to compute the diameter of the larger pancake graphs because of memory restriction. Hence, we changed the searching method to decrease the number of nodes drastically during the search process. In addition, by devising the representation of each node, we decreased the amount of the memory used. Moreover, we proved that distance computation is unnecessary in some cases and accelerated the search.

4.1 Depth-First Search

If we consider the process of computing diameters the tree search, the search method in the previous implementation corresponds to the breadth-first search inside a specific column in Figure 2. If we can replace it with the depth-first search, much memory space can be saved. However, the simple depth-first search will also search the vertices that have no relation to diameter computation.

Then, we used a method in which the vertices are judged if they have relation to diameter computation or not by using the incumbent diameter value. For a vertex π, if its upper bound value u is known, to judge if the vertex can be discarded or not, it is necessary to know to which column the vertex belongs in Figure 2. Then, let the number obtained by the following expression of $n = |\pi|$ and u be the column number in the figure of dependency relationship.

$$column = n \times 2 - u \tag{7}$$

If the column number calculated by substituting u with the incumbent diameter is less than or equal to the column number of the vertex which we are focusing on, we can discard the vertex.

If we perform the depth-first search by using this method, while the incumbent diameter is smaller than the true diameter, our implementation may search some vertices that are not searched by the previous implementation. However,

once the incumbent diameter becomes equal to the true diameter, this situation never occurs. Empirically, we can easily find the vertices that attains $f(n-1)+1$ during $f(n)$ computation. Hence, this method is efficient enough.

4.2 Elimination of Unnecessary Distance Computations

Up to now, we have computed the distance of the transformation even if it does not increase the upper bound, that is, $\sigma_n = T_n(\pi)$ for $\pi = \boldsymbol{e}^{\boldsymbol{x}}_{n-1}$. This transformation generates a permutation obtained by just adding n at the final position of the permutation $\pi = \boldsymbol{e}^{\boldsymbol{x}}_{n-1}$. However, it looks impossible to sort this kind of permutations with less operations than $f(\pi)$. Then, we guessed that $f(\pi) = f(\sigma_n)$ and proved it. Hence, there is no need to compute the diameter for $f(\sigma_n)$. By using this, we can improve A* search. By applying this improvement, we could accelerate the program by 5 to 8%.

Proof of $\boldsymbol{f(\pi) = f(\sigma_n)}$ Let $\pi = \boldsymbol{e}^{\boldsymbol{x}}_{n-1}$ and $\sigma_n = T_n(\pi)$, respectively. In general, to sort the permutation σ_n, it is necessary to execute multiple prefix reversals. Here, we abstract the operation sequence necessary to sort and denote it with an operation sequence $\boldsymbol{X}$.

First, we assume that there exists an operation sequence $\boldsymbol{X}$ for which $|\boldsymbol{X}| < f(\pi)$ holds. Then, let $\boldsymbol{Y} = (y_1, y_2, ..., y_m)$ be the operation sequence where y_i obtained by transforming each element x_i in $\boldsymbol{X} = (x_1, x_2, ..., x_m)$ as follows:

$$y_i = \begin{cases} x_i & x_i < n_{pos} \\ x_i - 1 & x_i \geq n_{pos} \end{cases} \tag{8}$$

where n_{pos} represents the position of n when the operations just before x_i are applied to σ_n.

Each y_i that is constructed by this transformation has the following features:

- In case that n is at the final position, it is just a reversal of no more than $n-1$ symbols.
- Order of the symbols except for n is same as that of the result of operation before transformation.

σ_n has n at the final position in the initial status. Therefore if we use $\boldsymbol{Y}$ instead of $\boldsymbol{X}$, we can sort π without performing the reversal of n symbols. In this case, the number of operations is $|\boldsymbol{Y}| = |\boldsymbol{X}|$. That is, if σ_n can be sorted by $|\boldsymbol{X}|$ operations, $f(\pi)$ can be also sorted by no more than $|\boldsymbol{X}|$ operations. This leads to contradiction. Hence, there does not exist $\boldsymbol{X}$ that satisfies $|\boldsymbol{X}| < f(\pi)$. From this, we can say $f(\pi) = f(\sigma_n)$.

Improvement of A* Search In the part of the diameter computation by A* search, one vertex which attains the least estimated distance is taken from enumerated elements, and the estimated distances for all of its neighbor vertices are computed. However, if the permutation corresponding to the vertex has

n in its final position, then from the proof above, we can see that the shortest distance is obtained without checking the vertex generated by the prefix reversal of n symbols. That is, we can see that there is a shortest path which does not include the vertex. Hence, in case that the final position has n, we can lessen the paths to be searched by ignoring the vertices obtained by the prefix reversal of n symbols. In addition, generalizing this idea, in case that the final part of the permutation is sorted, we can lessen more vertices to be searched by not operating them.

5 Parallelization

We have implemented the proposed system as a parallel system that works based on the Master-Worker method by using the MW framework[10]. By using MW, the number of Workers can be coped with automatically because Condor[11] performs the resource management. In addition, according to the function of MW, in case that some failures on the Worker side are detected, the executed tasks are migrated into normal Workers automatically.

Master fulfills the distribution of child problems and the collection of results, and Workers compute the given child problems. There is a variance among the sizes of child problems (the number of vertices for which distance computations are necessary) and the size of each child problem cannot be expected in advance. Therefore, if a Worker simply solves all of the child problems and returns the result, then it would be inefficient because the Worker that has finished its computation earlier must wait until the completion of computation of other Workers. Hence, we introduced a mechanism in which a Worker will suspend computation after a constant time and divide the suspended situation into multiple child problems. From this, we can maintain a constant number of tasks on the Master side all the time, and the Worker that has completed its computation can start its next computation immediately.

In addition, in parallel execution, we conducted a benchmark task in the initialization process of each Worker to measure the power of the machine on which the Worker runs. As the benchmark task, we selected the computation of $f(15)$. After a minute has passed, computation of the benchmark task on the Worker is stopped and we regard the number of vertices searched per second as the benchmark value of the Worker. Based on this value, we can estimate the execution time when we use other machines.

6 Computations of the Diameters of P_{16} and P_{17}

By using the implemented system, we actually computed the diameters of 16- and 17-pancake graphs. In both cases, we set both of the parameters in execution, the check pointing interval and the interval of the interruption of Worker computation to be 10 minutes. We also set the number of child problems which Master holds to be 1024. For these parameters, the optimal values are unknown. However, the values we set are proved to provide the sufficient performance based

Table 2. PC clusters configurations

Computation	Master/Worker	CPU	Memory	No. PCs	Connection
$f(16)$	Master	Pentium2 400MHz	256MB	1	100BASE-TX
	Worker	Pentium3 1GHz dual	1GB	16	
		Pentium2 400MHz	256MB	17	
$f(17)$	Master	Opteron 1.8GHz dual	2GB	1	1000BASE-T
	Worker	Opteron 1.8GHz dual	2GB	107	

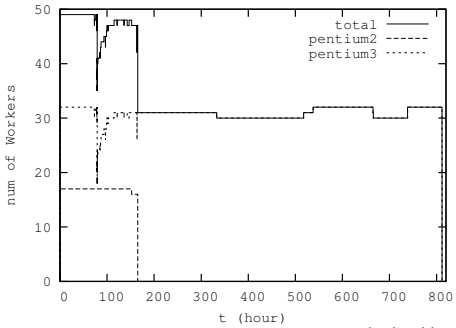

Fig. 3. Total number of Workers ($f(16)$)

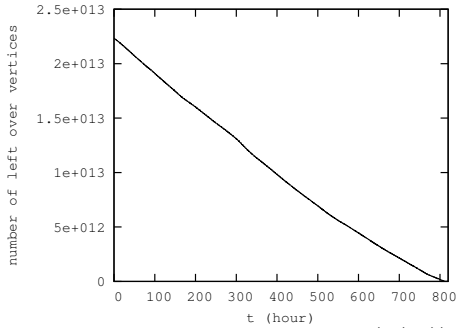

Fig. 4. Number of left over vertices ($f(16)$)

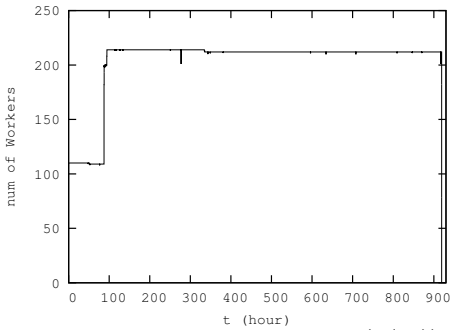

Fig. 5. Total number of Workers ($f(17)$)

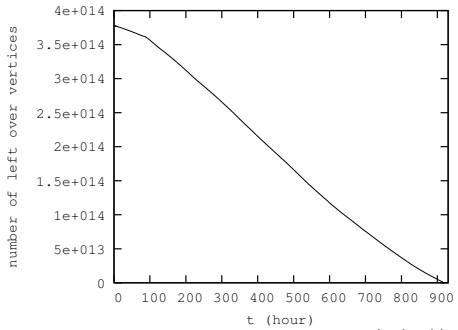

Fig. 6. Number of left over vertices ($f(17)$)

on preliminary experiments. Table 2 shows the PC clusters configurations that are used for the computations.

In computation of $f(16)$, by the computation during 33 days and 19 hours under the environment with 49 Workers at most, we have obtained the result $\boldsymbol{f(16) = 18}$. From expression (6), it is known that $f(16) \leq f(15) + 2 = 19$. Hence, we have checked that there is no vertex whose distance is 19. Figure 3 shows the change of the number of Workers in the process of computation. In this figure, the number of Workers of Pentium3 decreases rapidly around $t = 80$. This is because the MW framework found an ordinary user's job and a part of computation is automatically interrupted. In addition, around $t = 170$, Pentium2 machines are all stopped because of maintenance. Figure 4 shows the change of the number of remaining vertices in the process of the computation. Here, the number of vertices is the number in case all the vertices are assumed to be necessary for search. From this figure, we can see that the remaining vertices decrease almost linearly. Hence, we could find that the remaining computation time can be expected on the way of the computation process.

In computation of $f(17)$, by the computation during 38 days and 19 hours under the environment with 214 Workers at most, we have checked that there is no vertex whose distance is no less than 20 and obtained the result $\boldsymbol{f(17) = 19}$. Figure 5 shows the change of the number of Workers in the process of computation. In Figure 5, the number of Workers are rapidly increasing around $t = 90$, because we augmented the number of Workers assigned to the computation. The change of the numbers of remaining vertices is shown in Figure 6. Because the ratio of change varies around $t = 90$, we can see the effect of the augmentation of the assigned Workers.

We show some of the permutations and sorting sequences that attain the diameters in Table 3. The statistical information of the computations is shown in Table 4 where Overall Parallel Performance is a ratio of the total time of computation of Workers over the total working time of Workers. Though this value is ideally equal to 1, it is usually a smaller value practically, because of the overhead by communication and the unbalanced task granularity. However, in our system, the values are nearly equal to 1. Therefore we can see that it works very efficiently. We consider that this is because tasks are interrupted in constant time, and at least a constant number of tasks are maintained on the Master side all the time, hence all the Workers can execute the tasks all the time. In addition, Equivalent Run Time is the total sum of the multiplication of the execution time of each Worker and the benchmark value. This is the expected execution time when it is executed on the machine whose benchmark value is 1. The benchmark value of Pentium3 machine is about 1100 per one CPU. Hence, if all the Workers work all the time, then $f(16)$ can be computed in about 34 days in case of executing it on Pentium3 machines only. In addition, if we compute $f(17)$ by using 16 Pentium3 machines, which are used for computation of $f(16)$, then no less than 4 years would be necessary as the computation time.

Table 3. Examples of the permutations that attain the diameters of P_{16} and P_{17}

n	Permutation	Sorting Sequence
16	$(1, 15, 9, 11, 8, 10, 12, 7, 13, 5, 2, 16, 4, 14, 6, 3)$	$(10, 12, 16, 3, 5, 12, 3, 2, 4, 3, 5, 6, 8, 12, 3, 13, 15, 2)$
	$(6, 10, 4, 14, 2, 13, 16, 12, 8, 11, 7, 9, 5, 1, 3, 15)$	$(10, 8, 12, 5, 6, 2, 4, 14, 4, 15, 10, 2, 16, 15, 13, 2, 5, 3)$
	$(13, 9, 15, 2, 6, 4, 7, 11, 8, 12, 10, 14, 1, 16, 5, 3)$	$(11, 4, 3, 10, 6, 8, 9, 6, 13, 11, 14, 16, 3, 4, 2, 12, 14, 2)$
17	$(1, 4, 2, 7, 13, 3, 5, 17, 10, 15, 9, 14, 8, 12, 6, 16, 11)$	$(17, 8, 6, 10, 3, 8, 2, 12, 14, 3, 5, 8, 17, 2, 4, 3, 12, 6, 12)$
	$(7, 13, 2, 4, 1, 3, 5, 17, 10, 15, 9, 14, 8, 12, 6, 16, 11)$	$(12, 10, 2, 17, 10, 8, 12, 3, 10, 4, 5, 8, 17, 4, 3, 2, 12, 6, 12)$
	$(11, 15, 4, 2, 3, 1, 5, 8, 6, 17, 13, 16, 12, 14, 10, 7, 9)$	$(14, 7, 15, 16, 2, 7, 14, 12, 13, 11, 4, 12, 17, 6, 14, 4, 3, 2, 3)$

Table 4. Statistical information

n	16	17
Number of (different) workers	49	214
Wall clock time for this job (sec)	2921931.4774	3309757.6983
Overall Parallel Performance	0.9993	0.9994
Equivalent Run Time	103371746009.5473	2375697871296.6587

In this computation, we counted the number of discarded vertices as well as the number of searched to verify the correctness of the results of computation of the diameters. As a result, the sum of numbers of the discarded and the searched vertices matched the number of total vertices. Hence, we can conclude that the computation is correct. Since computation for each vertex is fulfilled in one CPU, we can also be fully confident in the correctness of the result of computation.

7 Conclusions

In this study, we have improved the method by Kounoike et *al.* to obtain the diameter of P_{15} so that it is applicable to compute the diameters of the larger scales of pancake graphs and implemented as a parallel computing system. In addition, we applied the system and obtained 16- and 17- pancake graphs by PC clusters. By conventional implementations, it has been impossible to compute the diameters of the larger pancake graphs because of memory restriction. However, our improved method can complete the computation if sufficient time is supplied and the computation time is shorten.

By using the implemented system, we have obtained the diameters of the pancake graphs up to $n = 17$. We want to obtain the diameters of larger pancake graphs. In addition, the known diameters so far satisfy $f(n) = f(n-1) + 2$ only when $n = 3, 6$ and 11 and no n has been found for $n > 11$ which satisfies the equation. We are also interested in such n's.

8 Acknowledgements

We would like to express our thanks to Prof. Mitsunori Miki and Prof. Tomoyuki Hiroyasu for permission to use the PC cluster on Doshisha University. A part of this research is supported by Japan society for the promotion of sciences, the grant-in-aid(No.16510105).

References

1. Dweighter, H. Amer. Math. Monthly **82** (1975) 1010
2. Akl, S.G., Qiu, K.: Fundamental algorithms for the star and pancake interconnection networks with applications to computational geometry. Networks **23** (1993) 215–225
3. Bass, D.W., Sudborough, I.H.: Pancake problems with restricted prefix reversals and some corresponding cayley networks. Journal of Parallel and Distributed Computing **63**(3) (2003) 327–336
4. Berthomé, P., Ferreira, A., Perennes, S.: Optimal information dissemination in star and pancake networks. IEEE Transactions on Parallel and Distributed Systems **7**(12) (1996) 1292–1300
5. Kumar, V., Grama, A., Gupta, A., Karypis, G.: Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjaming/Cummings (1994)
6. Quinn, M.J.: Parallel Computing: Theory and Practice, second edition. McGraw-Hill (1994)
7. Kounoike, Y., Kaneko, K., Shinano, Y.: Computing the diameters of 14- and 15-pancake graphs. In: Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks. (2005) 490–495
8. Heydari, M.H., Sudborough, I.H.: On the diameter of the pancake network. J. Algorithms **25**(1) (1997) 67–94
9. AT&T: On-Line Encyclopedia of Integer Sequences http://www.research.att.com/~njas/sequences/.
10. MW project: MW Homepage http://www.cs.wisc.edu/condor/mw/.
11. Condor Team: Condor Project Homepage http://www.cs.wisc.edu/condor/.