PC クラスタを用いた 16 パンケーキグラフの直径計算

浅井章吾[†] 鴻池祐輔[†]
品野勇治[†] 金子敬一[†]

n パンケーキグラフは，n 種類の記号で作られる順列をそれぞれ頂点とし，順列の前方反転によって移ることが可能な順列間を辺で結んだグラフである．n パンケーキグラフは $n!$ 個の頂点を持つので，頂点数に対する多項式時間のアルゴリズムでは，直径を求めることが困難な問題として知られている．これまでに，基本的な計算方法は提案されているが，実装面では $n = 15$ の規模の計算が限界であった．さらに大規模なパンケーキグラフの直径計算を行うためには，既存の計算方法における探索戦略を根本的に変更する必要がある．また，長時間の計算を可能とする十分なスケーラビリティを持つ並列システムとしての実装が不可欠である．本研究では，より大規模なパンケーキグラフの直径を求めるために改良した計算方法を提案する．また，計算方法は Conder/MW により並列システムとして実装し，PC クラスタを利用して実際に $n = 16$ の直径を求めた．

Computing the Diameters of 16-pancake Graph Using a PC Cluster

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

The n-pancake graph is a graph whose vertices are labeled by the permutations of n symbols. Two vertices are connected by an edge if and only if the corresponding permutations can be transitive by the prefix reverses. 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. A fundamental approach of the diameter computation has been proposed. However, the computation of the diameter of the n-pancake graph with $n = 15$ is the limit in practice. In order to compute the diameter of the larger pancake graphs, the search strategy in the diameter computation must be changed drastically. On top of that, it is indispensable to establish a sustainable parallel system with enough scalability. Therefore, in this study, we propose an improved algorithm to compute the diameter. We also have developed a sustainable parallel system with the Conder/MW framework, and computed the diameter of the 16-pancake graph by using a PC cluster.

1. はじめに

本稿では，全部が異なる大きさのパンケーキの山を，上に行くほど小さくなるようなパンケーキの山に並べ替えることをモデル化した問題を考える．このとき，並べ替えの操作としては，一番上から何枚かをまとめて反転させることだけが可能である．n 枚のパンケーキで作られる山のうち最悪の場合について，並べ替えるのに必要な反転の回数を n の関数として求める問題は**パンケーキ整列問題（*pancake sorting problem*）**[6] と呼ばれる．この問題は，順列の前方反転によって表されることから**前方反転問題（*prefix reversal problem*）**と呼ばれることもある．

パンケーキグラフ（*pancake graph*）は，1 から n までの n 種類の記号で作られる順列をそれぞれ頂点とし，順列の前方反転によって移ることが可能な順列の間を辺で結んだグラフである．n によって異なるグラフが作れることから，***n* パンケーキグラフ（*n-pancake graph*）**と呼ぶ．n パンケーキグラフは，$n!$ 個の頂点を持つ，$n-1$ 次の正則グラフである．パンケーキ整列問題とパンケーキグラフの直径を求める問題は等価な問題である．パンケーキグラフは，対称性，再帰性，次数と直径に対する頂点数の多さといった利点を多く持つことから，並列計算機システムにおける相互結合網のモデルとして注目されている[1),3),4)]．パンケーキグラフを相互結合網のモデルとして利用することを考えたとき，グラフの直径は通信遅延を示す尺度の 1 つとなる[8),10)]．

n パンケーキグラフの直径を求めるには，ある頂点

† 東京農工大学工学部情報工学科
Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology

表 1 n パンケーキグラフの既知の直径
Table 1 The diameters of n-pancake graphs.

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

から全頂点への最短距離を求めればよいが，頂点数や辺数に依存するアルゴリズムでは，計算時間と記憶容量が指数的に増加するため，すぐに現実的には解けなくなる．そこで，鴻池ら[11]はパンケーキグラフの再帰性に注目して，最短距離を求める必要がある頂点を限定する手法を提案した．この手法は Heydari ら[7]が 13 パンケーキグラフの直径を求めるときに使った手法を基本として，不要な探索を行わないように発展させたものである．それまで，$n \geq 14$ のパンケーキグラフの直径は未知であったが，鴻池らはこの手法を用いて $n = 14$，15 のパンケーキグラフの直径を与えた．パンケーキグラフの既知の直径を**表 1** に示す．数学的には，直径の数列そのものに対する関心も持たれており，*On-Line Encyclopedia of Integer Sequences*[2]には，$n = 13$ までの数列（数列名は Sorting by prefix reversal）が示されている．しかし，$n = 14$ 以上に関しては示されていない．より大規模なパンケーキグラフに対する直径を与えることは，このような数列の研究に対しても貢献する．

本研究では，鴻池らが 15 パンケーキグラフの直径を求めるときに用いた手法を，大規模なパンケーキグラフの直径計算が可能なように改良し，並列計算システムとして実装した．さらに，開発したシステムを利用し，これまでに求められていなかった 16 パンケーキグラフの直径を求めた．

2. 用語と記号の定義

ここでは，説明に必要な用語と記号を定義する．例を用いた詳細については文献 11) を参照されたい．

1 から n までの n 種類の記号で作られる順列の全体を集合 S_n とする．最小のパンケーキを記号 1，最大を記号 n として，一番上のパンケーキから順に対応する記号を並べた順列 $\pi \in S_n$ により n 枚のパンケーキの山を表す．整列された山に対応する順列 $(1, 2, \ldots, n)$ を $\boldsymbol{e}_n$ とする．順列 $\pi \in S_n$ の最初の k $(2 \leq k \leq n)$ 個の記号の順序を反転し，残りをそのままの順序とした順列を $\sigma \in S_n$ とすると，順列 π から順列 σ への変換を**順列 $\boldsymbol{\pi}$ に対する $\boldsymbol{k}$ 個の前方反転**という．また，$\pi^k = \sigma$ と表すこととする．本稿では順列の前方反転だけを扱うので，単に順列を k 個反転すると書いたときも順列を k 個前方反転することを意味する．

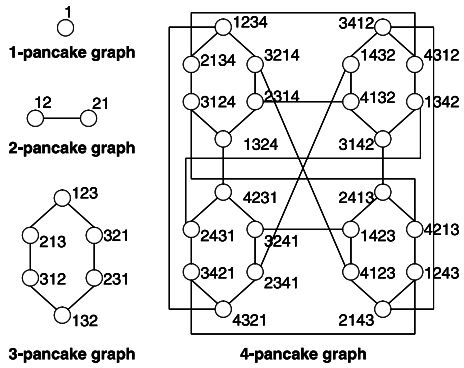

図 1 パンケーキグラフ
Fig. 1 The pancake graphs.

順列 π を x_1 個反転した結果をさらに x_2 個反転した結果に対応する $(\pi^{x_1})^{x_2}$ を $\pi^{(x_1,x_2)}$ のように表すこととする．さらに $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ とすると，$\pi^{\boldsymbol{x}}$ によって連続する $x_1, x_2, \ldots, x_m$ 個の反転を表すこととする．$\pi^{\boldsymbol{x}} = \boldsymbol{e}_n$ となるとき，$\boldsymbol{x}$ を π を整列する手順と呼ぶ．

与えられた順列 $\pi \in S_n$ を整列する手順の最小の反転回数を順列 π を引数とする関数 $f(\pi) = \min\{|\boldsymbol{x}| : \pi^{\boldsymbol{x}} = \boldsymbol{e}_n\}$ で表すこととする．n 枚のパンケーキの山を整列するのに必要な反転回数の最大を，n を引数とする関数 $f(n) = \max\{f(\pi) : \pi \in S_n\}$ で表す．慣習的に同じ関数 f が使われるが，引数によって意味が異なるので注意が必要である．

順列 $\pi \in S_n$ をそれぞれ頂点 π とし，$\sigma = \pi^k$ となる頂点 σ と頂点 π の間を辺で結んだグラフを**パンケーキグラフ**と呼ぶ．n によって異なるグラフが作れることから，それぞれのパンケーキグラフのことは **$\boldsymbol{n}$ パンケーキグラフ**と呼び，P_n で表す．実際の P_1〜P_4 を**図 1** に示す．

P_n の頂点で割り当てられた順列の最後の記号が j である頂点と，その頂点間を結ぶ P_n の辺を抜き出すことで作られる部分グラフを P_n^j と表すと，P_n^j は P_{n-1} と同型になる．このため，P_n^j のことを**部分パンケーキグラフ**と呼ぶ．各部分パンケーキグラフの頂点は重複しないから，P_n は n 個の部分パンケーキグラフと，部分パンケーキグラフをまたぐ辺からなる．この性質をパンケーキグラフの**再帰性**と呼ぶ．

1 から n までの n 種類の記号に対して 1 つの置換を決め，n パンケーキグラフの各頂点に割り当てられた順列をこの置換により書き換えたグラフは，元のパンケーキグラフと同型である．このことをパンケーキグラフの**対称性**という．

一般にグラフにおいて，ある頂点から別の頂点まで

の経路のうち，通過する辺の数が最も少ない経路を 2 頂点間の**最短経路**といい，その経路における辺の数を**最短距離**という．任意の 2 頂点間の組合せのうち，最も長い最短距離をグラフの**直径**という．

最短距離を調べる頂点の一方に $\boldsymbol{e}_n$ を選ぶことで，n パンケーキグラフの直径を求めることと，$f(n)$ を求めることは等価となる．本稿では，ある頂点 $\pi \in S_n$ と頂点 $\boldsymbol{e}_n$ との最短距離のことを単に π の距離と表すこととする．

3. 基本的な解法

直径を求めるための基本的な解法として，鴻池らが文献 11) で $f(15)$ を求めた手法を用いた．これは，パンケーキグラフの対称性・再帰性から頂点間の依存関係を求め，距離計算が必要な頂点を限定する方法である．

3.1 部分計算の依存関係

まず，順列 $\pi = \boldsymbol{e}_{n-1}^{\boldsymbol{x}} \in S_{n-1}$ について順列 $\sigma_k \in S_n$ $(1 \le k \le n)$ を式 (1) と定義する．このとき $f(\sigma_k)$ について式 (2) が成り立つ．

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

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

$\pi \in S_{n-1}$ に対し，$\sigma_k \in S_n$ を求める変換を $T_k(\pi)$ とし，集合 $S \subseteq S_{n-1}$ の要素すべてについて $T_k(\pi)$ を求めた集合を $T_k(S)$ とする．また，集合 S_n^m を $f(\pi) = m$ となる $\pi \in S_n$ の集合とし，$k < 0$ または $k > f(n)$ となる k については，S_n^k は空集合とする．このとき集合 $\overline{S}_n^m$ を

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

と定義する．これは，S_n のうち上界値が m である頂点集合である．

このとき式 (2) により $\pi \in \overline{S}_n^m$ について $f(\pi) \le m$ が成り立つ．また，$\overline{S}_n^m$ と S_n^m，S_n の間には以下の関係が成り立つ．

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

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

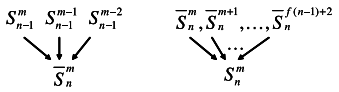

図 2 S_n^m と $\overline{S}_n^m$ との関係
Fig. 2 The relation between S_n^m and $\overline{S}_n^m$.

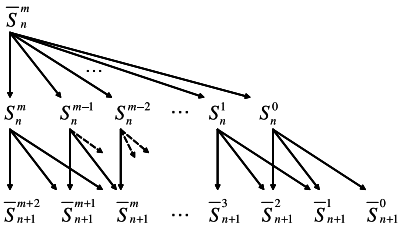

図 3 $\overline{S}_n^m$ と $\overline{S}_{n+1}^{x \le m+2}$ との関係
Fig. 3 The relation between $\overline{S}_n^m$ and $\overline{S}_{n+1}^{x \le m+2}$.

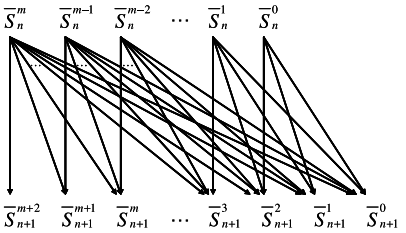

図 4 $\overline{S}_n^{x \le m}$ と $\overline{S}_{n+1}^{y \le m+2}$ との関係
Fig. 4 The relation between $\overline{S}_n^{x \le m}$ and $\overline{S}_{n+1}^{y \le m+2}$.

さらに，式 (4) から，

$$f(n) \le f(n-1) + 2 \quad (6)$$

ということが分かる．

式 (3) と式 (5) が表す関係を**図 2** に示す．この図では，下の集合の要素をすべて列挙するためには，上の集合が必要であることを意味している．また，これらの関係から $\overline{S}_n^m$ と $\overline{S}_{n+1}^x$ との関係を導き出したのが**図 3** である．このように，$\overline{S}_n^m$ は $\overline{S}_{n+1}^{x \le m+2}$ と関係していることが分かる．また，$\overline{S}_n^m$ から $\overline{S}_n^0$ までの関係も合わせて考え，中段を省略すると，**図 4** のようになる．さらに，この図の上下の関係も考えると，部分計算の依存関係は**図 5** のようになる．図 5 では，集合間の位置関係により，ある集合は，その真上および左上の集合すべてに依存していることを表している．また，図 5 の左下および右上の空白部分は空集合であることを表している．ある集合の下側が空集合かどうかは，直径が分かって初めて判断できる．このような関係によって，$S_1 = S_1^0 = \{\boldsymbol{e}_1\}$ から，変換と距離計算を再帰的に繰り返すことで任意の $\overline{S}_n^m$ を導くことができる．

3.2 直径の求め方

直径 $f(n)$ を求めるためには，まずすべての $\pi \in \overline{S}_n^{f(n-1)+2}$ について $f(\pi)$ を求める．このとき，$f(\pi) =$

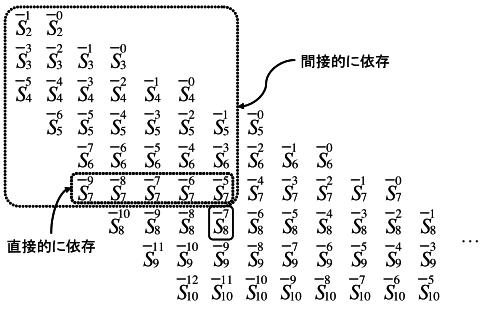

図 5 $\overline{S}_n^m$ 間の依存関係
Fig. 5 The dependency relation between $\overline{S}_n^m$.

$f(n-1)+2$ となるような π があるかないかによって $f(n)$ は以下のようになる．

- $f(\pi) = f(n-1)+2$ となる π がある場合：$f(n) = f(n-1)+2$ であり，このような π が見つかった時点で計算を終了できる（式 (6) 参照）．
- そのような π がない場合：$f(n) \leq f(n-1)+1$ であり，もしすでに $f(\pi) = f(n-1)+1$ となるような π が見つかっていれば，$f(n) = f(n-1)+1$ として計算を終了できる．そうでなければ $\pi \in \overline{S}_n^{f(n-1)+1}$ の中から $f(\pi) = f(n-1)+1$ となる π を探し，見つかれば $f(n) = f(n-1)+1$ である．

なお，個々の最短距離の求め方としては，A*探索を用いる．詳細については文献 11) を参照されたい．

3.3 既存の実装

鴻池らの実装では，図 5 において，左の列の集合の要素から順に変換と距離計算を行って集合の要素を確定させていき，その過程で直径を求める．この探索方法によって，集合間の依存関係から直径の計算に関係のないことが分かっている頂点を探索せずにすむ．しかし，パンケーキグラフの規模が大きくなるにつれ，集合の要素数が非常に多くなってしまい，メインメモリ上だけでは扱えなくなってしまう．また，後の計算に使えるように距離計算の結果をすべて保存するようになっている．これらの情報は指数的に増加し，$f(14)$ の計算終了時点では 21 GB もの圧縮ファイルをディスクに保存していた．このような実装であるため，直径計算が行える規模には限界がある．

4. 作成したシステムにおける実装

既存の実装では，メモリ使用量の問題により大規模なパンケーキグラフの直径計算を行うことはできない．そこで我々は探索方法を変えることによって，探索時に保持するノード数を大幅に削減した．加えて，各ノードの表現方法を工夫することにより，さらにメモリ使用量を削減した．また，距離計算が不要である場合があることを証明し，このことを用いて探索の高速化も実現した．

4.1 深さ優先探索

直径を求める過程を木構造の探索と考えると，既存の実装の探索方法は図 5 の特定の列内での幅優先探索であると考えられる．これを深さ優先探索に変えることができればメモリ使用量を大幅に減らすことができる．しかし，単純に深さ優先探索を行うだけでは，直径計算に関係しない頂点も探索してしまうことになる．

そこで，暫定直径の値を使って直径計算に関係しない頂点を判断する方法を用いた．ある頂点 π と，その上界値 u が分かっているとき，実際にその頂点を破棄してよいかを判断するためには，図 5 のような依存関係の図において，その頂点がどの列に属しているか知る必要がある．そこで，$n = |\pi|$ と u から次の式によって得られる数を，依存関係の図上での列番号とする．

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

上式の u の部分に暫定直径を入れて算出した列番号が，注目している頂点の列番号と等しいか小さい場合は，その頂点を破棄することができる．

この手法を用いて深さ優先探索を行うと，暫定直径が実際の直径より小さい間は既存の実装では探索されない頂点を探索しうるが，暫定直径が実際の直径と等しくなると，このようなことはなくなる．経験的に，$f(n)$ を求めるときに $f(n-1)+1$ となる頂点はすぐに見つかることが分かっているので，この方法はきわめて効率的だといえる．

4.2 遅延評価

探索は頂点と上界値の組から，新しい頂点と上界値の組を作り出していくことで行う．そこで，頂点と上界値の組合せを子問題と呼び，これを $[\pi, u]$ と表現する．これを内部的には，$[\pi, f(\pi), k]$ という形で扱い，必要になった時点で，$[\sigma_k, f(\pi)+\alpha]$ と置き換えるようにした．σ_k，α は式 (1)，(2) より求める．つまり，σ_k の算出を遅延させている．$[\pi, u]$ という表現を用いた場合の Pool（深さ優先探索におけるスタック）とのやりとりを**図 6** に示す．また，$[\pi, f(\pi), k]$ という表現を用いた場合の Pool とのやりとりを**図 7** に示す．k の値は，式 (8) に従って $k = 2$ から順次変えて，$k = |\pi|+1$ となったら，次回の操作時に破棄する．ここで式 (8) は，式 (2) で求められる上界値が大きいものから出てくるように定めた．

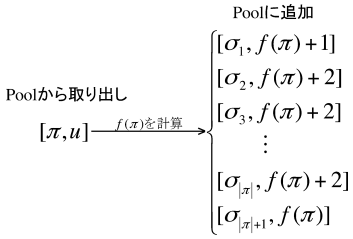

図 6 即時評価
Fig. 6 Eager evaluation.

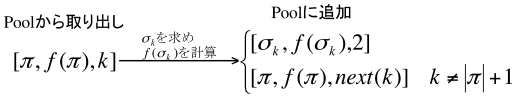

図 7 遅延評価
Fig. 7 Lazy evaluation.

$$next(k) = \begin{cases} k+1 & 2 \le k < |\pi| \\ 1 & k = |\pi| \\ |\pi|+1 & k = 1 \end{cases} \quad (8)$$

この表現を用いると，k を変化させることで別の頂点を表すことができる．これによって，メモリ使用量，メモリの確保・開放にかかる時間，Pool への出し入れにかかる時間を減らすことができる．

4.3 不要な距離計算の削除

いままでは上界値が増えない変換（$\pi = \boldsymbol{e}^{\boldsymbol{x}}_{n-1}$ に対する $\sigma_n = T_n(\pi)$）についても距離計算をしていた．この変換は順列 $\pi = \boldsymbol{e}^{\boldsymbol{x}}_{n-1}$ の最後尾に n を追加しただけの順列を作成するものである．しかし，このような順列が $f(\pi)$ より短い手順で整列できるとは考えにくい．そこで我々は，$f(\pi) = f(\sigma_n)$ なのではないかと考え，これを証明した．したがって，$f(\sigma_n)$ については距離計算をする必要がない．また，このことを用いて A*探索を改良することができる．これらの，改良を施すことでプログラムを 5〜8%ほど高速化することができた．

4.3.1 $f(\pi) = f(\sigma_n)$ の証明

$\pi = \boldsymbol{e}^{\boldsymbol{x}}_{n-1}$，$\sigma_n = T_n(\pi)$ とする．一般に，順列 σ_n を整列するには，複数回の前方反転を行う必要がある．ここでは，整列に必要な操作列を抽象化し，**図 8** のように，操作列 $\boldsymbol{X}$ として表す．

まず，$|\boldsymbol{X}| < f(\pi)$ となる操作列 $\boldsymbol{X}$ が存在すると仮定する．ここで，$\boldsymbol{X} = (x_1, x_2, \ldots, x_m)$ の各要素 x_i を次のように変換して得られる y_i からなる操作列を $\boldsymbol{Y} = (y_1, y_2, \ldots, y_m)$ とする．

$$y_i = \begin{cases} x_i & x_i < n_{pos} \\ x_i - 1 & x_i \ge n_{pos} \end{cases} \quad (9)$$

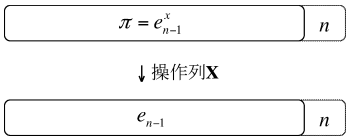

図 8 σ_n の整列操作
Fig. 8 Sorting σ_n.

ただし，n_{pos} は σ_n に対して x_i の直前の操作まで行ったときの n の位置である．

この変換によって作られる y_i には以下のような特徴がある．

- n が最後尾にあるときには，$n-1$ 個以下の反転操作にしかならない．
- n を除いた順序関係が変換前の操作の結果と同じである．

σ_n は初めの状態で n が最後尾にある．したがって，$\boldsymbol{X}$ の代わりに $\boldsymbol{Y}$ を用いれば，n 個反転という操作をせずに π を整列できる．このときの操作回数は $|\boldsymbol{Y}| = |\boldsymbol{X}|$ である．つまり，$|\boldsymbol{X}|$ 回の操作で σ_n が整列できるなら同等以下の操作回数で $f(\pi)$ が整列可能ということになり，これは仮定と矛盾する．したがって，$|\boldsymbol{X}| < f(\pi)$ となる $\boldsymbol{X}$ は存在しない．このことから，$f(\pi) = f(\sigma_n)$ だといえる．

4.3.2 A*探索の改良

A*探索による距離計算部分では，列挙された要素の中から，距離の見積りが最も小さい頂点を 1 つ取り出し，その頂点に隣接するすべての頂点について距離の見積りを計算する．しかし，この頂点が表す順列 π（$|\pi| = n$）の最後尾が n の場合，上記の証明から，n 個前方反転した頂点を調べなくても，最短距離を求めることができる（その頂点を通らない最短経路が存在する）ことが分かる．したがって，最後尾が n の場合に n 個前方反転した頂点は調べないことにすると探索すべき経路を減らすことができる．また，この考えを一般化し，順列の末尾が整列されているならば，そこには操作をしないようにすると，探索すべき頂点をさらに減らすことができる．

4.4 改善結果

上記の変更を施したプログラムの速度を既存の実装と比較した．その結果を**表 2** に示す．プログラムの実行には，Pentium 4 2.6C GHz，1 GB Dual Channel DDR-SDRAM の計算機を用いた．この計算機は文献 11) で鴻池らが，実際の実験に使用した計算機である．また，既存の実装とは探索方法が異なるので，単純に実行時間を比較することはできない．なぜなら，既存の実装では，以前の直径計算の結果を利用して探

表 2 既存の実装との速度比較
Table 2 Speed comparison with existing implementation.

n	速度向上
8	50.00%
9	59.43%
10	50.00%
11	93.19%
12	35.10%
13	21.62%

索を行うため，直径が 2 増えた直後（$n = 12$ の場合など）の実行時間が極端に短いからである．そこで表 2 では，既存の実装で $f(n)$ を求めるのに要する時間を，すべての $\{f(m) : 2 \leq m \leq n\}$ を求めるのに要する時間の総和としている．なお，このような方法をとらずに比較した場合でも，$n = 12$ の部分を除けば表 2 と同じような傾向が見られる．このような結果から，既存の実装よりも効率の良い探索が行えているといえる．

5. 並 列 化

提案した手法を MW フレームワーク[9] を用いて Master-Worker 方式で動作する並列システムとして実装した．MW を用いることで，資源管理は Conder[5] が行うため Worker 数の増減にも自動的に対処できる．また MW の機能により，Worker 側に障害が検知された場合に，実行されていたタスクは正常な Worker に自動的に移される．

Master が子問題の配布や結果の集計を行い，Worker は与えられた子問題の計算を行う．各子問題の規模（距離計算が必要な頂点数）にはばらつきがあり，各子問題の規模は実際に探索が終わるまで分からない．そのため，単純に子問題をすべて解いてから結果を返していたのでは，早く計算が完了した Worker が他の Worker の計算完了を待つということになってしまい効率が悪い．そのため，Worker は一定時間で計算を中断し，中断した状態から複数の子問題に分割するようにした．これによってつねに Master 側にタスクを一定数確保することができるため，計算が終了した Worker は即座に次の計算を始めることができる．

また，並列実行の際に各 Worker の能力を計測するために，Worker 初期化時にベンチマークタスクを実行することとした．ベンチマークタスクとしては，$f(15)$ を求める計算を実行させる．1 分経過したところで計算を中断し，1 秒あたりいくつの頂点を探索したかをベンチマークの値とする．この値をもとに，別の計算機を使用した場合の実行時間などを推測することができる．

表 3 実験に用いた計算機
Table 3 Machines.

Master/Worker	CPU	Memory	台数
Master	Pentium2 400 MHz	256 MB	1
Worker	Pentium3 1 GHz dual	1 GB	16
	Pentium2 400 MHz	256 MB	17

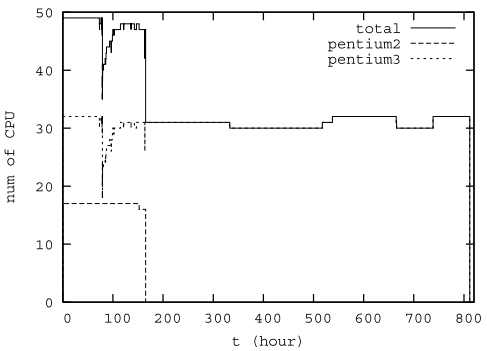

図 9 Worker の総数の変化
Fig. 9 The total number of workers.

6. 計 算 実 験

作成したシステムを用いて，実際に 16 パンケーキグラフの直径を計算した．実験に用いた計算機を**表 3** に示す．これらの計算機が 100BASE-TX のイーサネットで接続された環境で実験を行った．なお，実行のパラメータとして，チェックポイント間隔および Worker の計算を中断する時間を 10 分，Master が保持する子問題数を 1,024 個とした．パラメータについては，最適な値は分かっていないが，これらの値で十分な性能が得られることが予備実験によって分かっている．最大 49 の Worker による 33 日と 19 時間に及ぶ計算の結果，$\boldsymbol{f(16) = 18}$ という結果を世界で初めて求めた．式 (6) から $f(16) \leq f(15) + 2 = 19$ ということが分かっていたので，この計算において距離が 19 となる頂点がないことが確認できたといえる．

$f(16)$ 計算中の Worker 数の変化を**図 9** に示す．この図において，80 時間付近で Pentium3 の Worker 数が急激に減っている．これは，MW フレームワークが一般ユーザのジョブ実行を検出し，一部の計算が自動的に中断されたためである．また，170 時間付近でメンテナンスの都合により，Pentium2 のマシンをすべて止めている．

計算中の残りノード数の変化を**図 10** に示す．ここでのノード数とは，すべての頂点が探索に必要だと仮定した場合の数である．この図から，残りノード数はほぼ線形に減少していることが分かる．したがって，

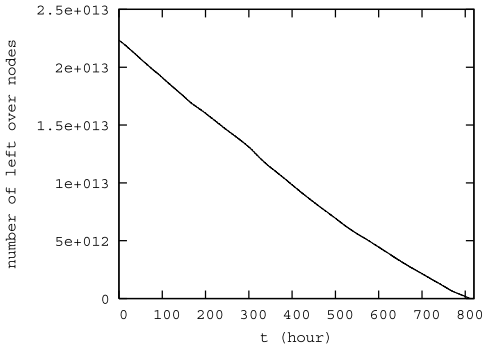

図 10 残りノード数の変化
Fig. 10 The number of left over nodes.

表 4 統計的な情報
Table 4 Statistical informations.

Number of (different) workers	49
Wall clock time for this job (sec)	2921931.4774
Overall Parallel Performance	0.9993
Equivalent Run Time	103371746009.5473

計算途中において残りの計算時間を予測できることが分かった．

実験の統計的な情報を**表 4** に示す．ここで Overall Parallel Performance とは，Worker が計算をしていた時間の総和を Worker の稼働時間の総和で割ったものである．この値は理想的には 1 であるが，実際は通信などによるオーバヘッドや，タスク粒度の偏りによって小さい値になってしまうことが多い．しかし，本システムでは 0.9993 となっており，非常に効率的に動作していることが分かる．これは，一定時間でタスクを中断し，Master 側につねにタスクを一定数確保するようにしているため，つねにすべての Worker がタスクを実行できるためであると考えられる．また，Equivalent Run Time とは，各 Worker の実行時間とベンチマーク値の積の総和である．これは，ベンチマーク値が 1 のマシンで実行した場合に予想される実行時間であると考えることができる．Pentium3 のマシンのベンチマーク値が 1CPU あたり約 1100 であったので，Pentium3 のマシンだけで実行した場合でも，すべての Worker がつねに動いていれば約 34 日で計算できたことが分かる．

作成したシステムで直径を求めた際に，実際に探索を行った頂点数を**表 5** に示す．比較のために，鴻池らの実装での値もあわせて示す．鴻池らの実装では，$f(n)$ を求める際に $f(n-1)$ までの計算結果を利用できるようになっていた．そのため，鴻池らの実装における $f(1)$〜$f(n)$ で探索を行った頂点数の総和が，本システムにおける $f(n)$ で探索を行った頂点数に対応

表 5 探索を行った頂点数
Table 5 The number of vertices.

n	探索を行った頂点数（旧）	探索を行った頂点数（新）	P_n の頂点数
14	249,271,566	269,136,281	87,178,291,200
15	3,640,943,222	3,910,001,456	1,307,674,368,000
16	—	60,439,519,614	20,922,789,888,000

することに注意されたい．また，鴻池らの実装では集合の全要素が確定した $\overline{S}_n^m$ から距離計算を行っていくため，不要な列挙がまったく行われない．これらのことから，本システムの頂点数の方が若干多くなっている．本システムにおける頂点数は，実行ごとに変化するが，$f(16)$ を求めたときでも，暫定直径がただちに更新され，不要な列挙はきわめて少なく抑えられた．このことからも，本システムは非常に効率的に並列化が行えたといえる．

また，直径計算の結果の正しさを検証するために，探索した頂点数のほかに，破棄した頂点数もカウントしていた．その結果，探索終了時点で探索した頂点数と破棄した頂点数の和が全頂点数と一致した．したがって，全頂点に対して検証が行われたといえる．各頂点に対する計算は 1CPU 内の処理で完結しており，計算の結果は十分信頼できるものだといえる．

実際に 18 回の反転が必要であった長さ 16 の順列をいくつか示す．

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

これらの順列はそれぞれ次の手順により，18 回の前方反転で整列することができる．

$(10, 12, 16, 3, 5, 12, 3, 2, 4, 3, 5, 6, 8, 12, 3, 13, 15, 2)$
$(10, 8, 12, 5, 6, 2, 4, 14, 4, 15, 10, 2, 16, 15, 13, 2, 5, 3)$
$(11, 4, 3, 10, 6, 8, 9, 6, 13, 11, 14, 16, 3, 4, 2, 12, 14, 2)$

7. おわりに

本研究では，鴻池らが P_{15} の直径を求めるときに用いた手法を，大規模なパンケーキグラフの直径計算が可能なように改良し，並列計算システムとして実装した．また，実際にこのシステムにより，PC クラスタを用いて 16 パンケーキグラフの直径を求めた．既存の実装では記憶容量の点において大規模なパンケーキグラフの直径計算を行うことができなかったが，改良により時間さえあれば計算を行えるようになり，計算時間も短縮することができた．

作成したシステムによって，$n = 16$ までのパンケーキグラフの直径を求めたが，より大きな n について

も直径を求めたいと考えている．これは，計算量の増加から考えると，現在我々が所有している計算機環境では不十分である．しかし，PC の高性能化・低価格化を考えると，$f(17)$ を求めるのは計算機環境を整えて，あるいは借用することで十分可能である．たとえば，dual Opteron（1.8 GHz）のマシンが 50 台（100CPU）ある環境を考える．この環境ではベンチマーク値が 1CPU あたり約 3550 となることが分かっている．表 4 の Equivalent Run Time から，$f(16)$ の計算は約 3.4 日で終わることが分かる．また $f(17)$ の計算は，必要な計算量が単純に 17 倍になったと仮定した場合，約 57.3 日で計算が可能であると予測できる．このように，環境さえ整えれば $n \geq 17$ における直径も求めることができる．さらに，利用環境における実行時間も予測可能となった．

また，既知の直径は，$n = 3, 6, 11$ のときだけ $f(n) = f(n-1)+2$ となっている．$n > 11$ では，まだこのような n は見つかっていない．このような n が，どこで出てくるかということにも興味がある．

参 考 文 献

1) Akl, S.G. and Qiu, K.: Fundamental Algorithms for the Star and Pancake Interconnection Networks with Applications to Computational Geometry, *Networks*, Vol.23, pp.215–225 (1993).
2) AT&T: On-Line Encyclopedia of Integer Sequences. http://www.research.att.com/~njas/sequences/
3) Bass, D.W. and Sudborough, I.H.: Pancake Problems with Restricted Prefix Reversals and Some Corresponding Cayley Networks, *Journal of Parallel and Distributed Computing*, Vol.63, No.3, pp.327–336 (2003).
4) Berthomé, P., Ferreira, A. and Perennes, S.: Optimal Information Dissemination in Star and Pancake Networks, *IEEE Trans. Parallel and Distributed Systems*, Vol.7, No.12, pp.1292–1300 (1996).
5) Condor Team: Condor Project Homepage. http://www.cs.wisc.edu/condor/
6) Dweighter, H.: *Amer. Math. Monthly*, Vol.82, p.1010 (1975).
7) Heydari, M.H. and Sudborough, I.H.: On the diameter of the pancake network, *J. Algorithms*, Vol.25, No.1, pp.67–94 (1997).
8) Kumar, V., Grama, A., Gupta, A. and Karypis, G.: *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjaming/Cummings (1994).
9) MW project: MW Homepage. http://www.cs.wisc.edu/condor/mw/
10) Quinn, M.J.: *Parallel Computing: Theory and Practice*, second edition, McGraw-Hill (1994).
11) 鴻池祐輔，金子敬一，品野勇治：パンケーキグラフの直径計算，情報処理学会論文誌：数理モデル化と応用，Vol.46, No.SIG10 (TOM12), pp.48–56 (2005).

(平成 17 年 8 月 25 日受付)
(平成 17 年 10 月 13 日再受付)
(平成 17 年 11 月 8 日採録)

浅井 章吾

1983 年生．2003 年国立群馬工業高等専門学校電子情報工学科卒業．2005 年東京農工大学情報コミュニケーション工学科卒業．現在，同大学大学院工学府情報コミュニケーション工学専攻在学中．

鴻池 祐輔（正会員）

1978 年生．2006 年東京農工大学大学院工学教育部電子情報工学専攻博士後期課程修了，博士（工学）．同年東京農工大学工学府・工学部産学官連携研究員（現職）．主に整数計画問題の解法とその並列化に興味を持つ．オペレーションズ・リサーチ学会会員．

品野 勇治（正会員）

1961 年生．1997 年東京理科大学大学院工学研究科博士課程修了，博士（工学）．同年東京理科大学助手．1999 年東京農工大学講師，2004 年同大学助教授（現職）．主に，数理計画法の理論と応用の研究，組合せ最適化問題に対する並列・分散アルゴリズムとその実装に関する研究に従事．オペレーションズ・リサーチ学会，IEEE，ACM 各会員．

金子 敬一（正会員）

1962 年生．1985 年東京大学工学部計数工学科卒業．1987 年同大学大学院工学系研究科情報工学修士課程修了．同大学計数工学科助手，千葉大学情報工学科講師を経て，現在，東京農工大学情報工学科助教授．博士（工学）．関数プログラミング，ディペンダブルコンピューティング，マルチメディア教育等の研究に従事．日本ソフトウェア科学会，電子情報通信学会，ACM 各会員．