フラッシュメモリファイルシステムにおけるメモリ割当ての効果

最所圭三[†] 福田晃[††]

フラッシュメモリは，不揮発性，高記録密度，低消費電力という特徴から，多くの小型携帯用の組み込み機器の記憶デバイスとして採用されている．しかし，フラッシュメモリはDRAMのような揮発性半導体メモリと比較してデータ書き込みが遅く，データ書き換えの際には時間のかかるブロック消去を必要とする．さらに，ブロック消去は，回数の制限があるため，できる限り避けなければならない，という問題点を持つ．このため，リンク構造を持つファイルシステムを提案し，評価を行った．その結果，提案したファイルシステムはブロック指向のファイルシステムに比べ，ブロック消去が分散することを確認した．しかし，評価を重ねることにより，同じ量の書き込みであっても，利用環境によって実際に行われる書き込み量が大きく異なることが分かった．特に，複数のファイルを同時に書き込む場合の，実際の書き込み量が多くなることが分かった．本論文では，提案しているファイルシステムに対して，種々のパラメータを用いて行った評価結果を報告する．さらに，その結果に基づいてメモリ割当て手法を改良し，その評価を行う．その結果，複数のファイルを同時に書き込む場合でも，単独でファイルを書き込む場合と同等の実際の書き込み量になった．

Effect of Memory Allocation on Flash Memory File System

Keizo Saisho[†] and Akira Fukuda[††]

A flash memory device is often used in portable embedded systems as a non-volatile storage device because it is non-volatile, high density and low power memory device. The write access time of flash memory devices is, however, longer than that of DRAM and SRAM. Moreover, the block erase operation, which must be performed before data modification, spends much time, and the number of it is limited. Therefore, the authors proposed a linked list file system and evaluated it. As the result, the proposed file system was confirmed that it dispersed block erase operations more than block oriented file systems. The results also indicated that the amount of real write size varied by used environment even if file size is same. Specially, the amount of real write size increased when multiple files were created concurrently. This paper reports the evaluation of proposed file system with several parameters, improves memory allocation algorithm, and evaluates improved file system. As the result, real write size with concurrent write operations is nearly equal to that with single write operation.

1. はじめに

フラッシュメモリは，不揮発性，高記録密度，低消費電力という特徴を持つ．さらに，ディスクドライブを比較すると振動に強く小型軽量である．このため，デジタルカメラや携帯電話など，小型の携帯型組み込み機器の記憶デバイスとしての採用が増えてきている[1)]．しかし，フラッシュメモリは，以下に示す問題点を持っている[2),3)]．通常の書き込みでは，データの各ビットを同じ状態のままにするか，初期状態から他の状態にしか変更できない，という方向性を持っている．たとえば，初期状態が‘1’のデバイスでは，‘0’から‘0’，‘1’から‘1’および‘1’から‘0’には遷移できるが，‘0’から‘1’には遷移できない．逆方向に変更するためには，ブロック消去という処理が必要であるが，この処理は消去ブロックと呼ぶ数KBから数10KB単位で行われるため，消去ブロックの一部を消去したい場合は，1) 他の領域をRAMにコピーする，2) ブロック消去を行う，3) コピーしていた内容を書き戻す，の3つ操作が必要である．ブロック消去は，1秒程度の時間を要し，書き込みは5〜10μ秒というDRAMなど従来の揮発性半導体メモリと比較して100倍近くの時間を要するため，データの更新は非常にコストのかかる処理になる．さらに，ブロック消去の回数には制限があるため，できる限り避けなければならない．

このフラッシュメモリを用いて不揮発性の主記憶を

† 香川大学
Kagawa University
†† 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

実現する研究[4]も行われているが，一般には，その不揮発性と書き込みオーバヘッドのために 2 次記憶として扱い，その上にファイルシステムを構築して用いている．磁気ディスクを対象とした一般的なファイルシステムでは，ファイル情報を持つ領域が固定領域に置かれる．さらに，ファイルの作成，更新，削除などの操作を行うたびに，ファイル情報を持つ領域の書き換えが発生する．フラッシュメモリでは，ブロックの内容を書き換える際にブロック消去を必要とするため，このファイルシステムをそのまま実装すると，ファイル情報を持つ領域を置いているブロックに対するブロック消去が頻発する．ブロック消去の回数には限界があり，この限界を超えてブロック消去を行った場合，フラッシュメモリの正しい読み書きは保証されない．したがって，このブロック消去の限界を考慮しないファイルシステムでは，フラッシュメモリの一部のブロックがブロック消去の限度回数を超えて消去され，フラッシュメモリ全体が正常に使用できなくなってしまうという問題点を持つ．

このため，我々は，書き込みの方向性に対処するためにリンク構造のファイルシステム lffs（Linked list Flash memory File System）を提案している[5]．lffs では，すべての操作を追記で行うことにより，書き込みの方向性に対処している．たとえば，ファイル名の変更は，新しいファイル名をリンクに接続し，元のファイル名を無効にするフラグを立てることにより実現する．この結果，ブロック消去が一部のブロックに集中することはなくなった．また，追記を行うことにより，ファイルの大きさが消去ブロックの大きさよりかなり小さい場合でも，使用効率がほとんど落ちなかった．

しかし，種々の条件で実験を行うことにより，lffs は以下の問題を持つことが分かった．

(1) メモリの使用率が高くなると，同じ書き込み量であっても実際の書き込み量が多くなる．
ここでいう，メモリの使用率とは，メモリ全体に対する，全ファイルの有効なデータが占める割合である．

(2) 複数のファイルに対して同時に書き込むと，実際の書き込み量が多くなる．

(3) 書き込む単位が小さくなると，実際の書き込み量が多くなる．

(1) の問題は，どのファイルシステムでも共通の問題である．フラッシュメモリを用いる場合，あるデータを書き換える場合，対象のデータが含まれる消去ブロック中の有効なデータの読み出し，消去ブロックの消去，消去ブロックへの有効なデータの書き戻し（ガベージコレクション）をともなう．この場合，各ブロックにおいて，有効なデータが平均メモリの使用率で占めているため，新たに使用できる領域は残りの領域になる．このため，メモリの使用率が高くなると，回収できるメモリ量が減り，同じ量の書き込みであっても，ガベージコレクションが増え，ガベージコレクションにともなう書き込みが加わる．

複数のファイルの書き込みが同時に発生すると，1 つの消去ブロックにそれらのファイルが配置されるため，1 つのファイルが，あるブロックに占める割合が小さくなる．これにより，ファイルが消去されて発生する空き領域が分散するため，ガベージコレクションで回収できるメモリ量が少なくなり，(2) の現象が発生する．また，書き込む単位が小さくなると，各リンクのヘッダのオーバヘッドが大きくなるだけでなく，(2) の原因になる 1 つのファイルが，分散して配置される傾向がより強くなり，(3) の現象が発生する．

このため，本論文では，ファイルを書き込み用に開く場合に，専用のブロックをそのファイルに割り当てるようにすることで，(2)，(3) の問題点を解決する．このため，ガベージコレクションの契機を，使用できるメモリがなくなった時点だけでなく，ファイルを書き込み用に開く場合も加えた．これにより，同時にファイルを書き込む場合だけでなく，書き込みの単位が小さい場合でも，改良前の単一の書き込みの場合と同等の書き込み量で抑えられた．

ここで得られた結果は，lffs 固有の問題でなく，フラッシュメモリを使用するファイルシステムの共通の問題であり，ここで提案した手法の概念は，他のファイルシステムにも適用できるものである．

以下，2 章で提案しているリンク構造ファイルシステム lffs の構造を示し，3 章で種々のパラメータで lffs を評価する．4 章で 3 章の結果に基づき，lffs のメモリ割当てアルゴリズムを改良し，5 章でその評価を行う．

2. リンク構造ファイルシステム lffs

本章では文献 5) で提案した lffs について説明する．

2.1 lffs の設計方針

フラッシュメモリにディレクトリなど，ファイル情報を持つ領域が固定領域に置かれるファイルシステムを構築すると，“管理情報が割り当てられた消去ブロック” へのブロック消去が集中して，すぐに更新回数の上限に達してしまう．その結果，他の消去ブロックの消去回数が限界未満であっても，そのフラッシュメモリを使用できなくなる．

これに対処するために，我々は，**図 1** に示す，リ

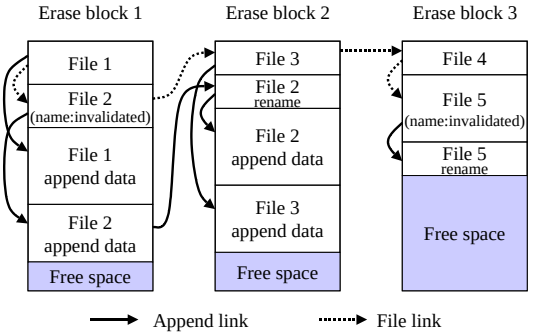


図 1 リンク構造のファイルシステム
Fig. 1 Link structured file system.

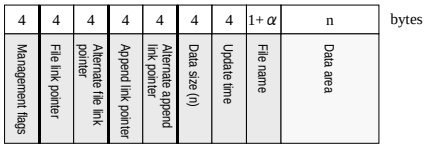


図 2 ファイルエントリの基本構成要素
Fig. 2 Elements of basic file entry.

図 3 管理フラグの構成要素
Fig. 3 Elements in management flags.

ンク構造のファイルシステム lffs（Linked list Flash memory File System）を提案した．リンクで接続されている各領域を，本論文ではファイルエントリと呼ぶ．図中のファイルリンクは，ファイル間をつなぐためのリンクで，追記リンクは，ファイル内のファイルエントリをつなぐためのリンクである．lffs では，すべての操作を追記で行うことにより，書き込みの方向性に対処している．たとえば，ファイル名の変更は，図の File 2 や File 5 のように，新しいファイル名を持つファイルエントリをリンクに接続し，元のファイル名を無効にすることにより実現する．この場合，フラグは 1 回だけ状態を変更できるので，初期値を有効に，変更した状態を無効に割り当てる．また，ファイルの削除は，削除マークを付け，その部分を無視することで実現する．これにより，ブロック消去が一部のブロックに集中することがなくなる．しかし，リンク構造を用いるので，2 つの問題が発生する．1 つは，ファイルの作成・削除を繰り返すうちに，使用できるメモリがなくなることである．もう 1 つは，書き込みが行われたリンクポインタを変更できないことである．前者はフラッシュメモリでなくても生じる問題であり，ガベージコレクションを導入することで対処する．後者はフラッシュメモリ特有の問題である．ファイルの削除や更新によりファイルエントリのリンクポインタを更新しなければならないが，更新のためにはブロック消去を必要とする．このため，本ファイルシステムでは，“予備リンクポインタ” および “削除ビット” を導入する．しかし，これだけでは，変更が 1 回に限定されるので，リンク情報のみで構成されるリンク書き換えエントリを導入する．

2.2 lffs の設計

本節では，lffs の特徴であるファイルエントリの構造，リンク書き換えエントリ，およびガベージコレクションについて説明する．詳しくは文献 5) を参照のこと．

- ファイルエントリの構造
  lffs では，ファイルエントリをリンクリストで接続することにより，ファイルの属性やデータを更新する．このため，ファイルエントリには，**図 2** に示す要素を含める．すべてのファイルエントリがすべての要素を含むわけではない．たとえば，データを追記する場合，ファイル名やファイルリンクポインタは必要ではない．このため，管理フラグに**図 3** に示す要素を含める．フラグ内の各要素は 1 ビットで構成される．フラグがセットされたときに図中の状態を表すようにする．このため，いったんフラグがセットされると，逆の状態に戻せない．Include で始まるビットは，対象の要素が含まれるかを示し，Invalid で始まるビットは，対象の領域が有効であるかを示す．Use から始まるビットは，対象のポインタを使用しているかを示す．なお，予備リンクポインタを使用しているときは，主リンクポインタは自動的に無効化される．
- リンク書き換えエントリ
  リンク書き換えエントリは，リンクポインタのみからなるファイルエントリである．**図 4** に示すようにファイルブロック FB2 が消去される場合，リンク書き換えエントリ LM を生成し，FB1 の予備リンクポインタが LM を指すようにする．さらに，LM のリンクポインタが FB3 を指すようにすることで，ファイルリンクを再構成する．LM で指されているファイルエントリを削除する場合は，図中の点線で示すようにさらに LM2 を生成し，LM2 のリンクポインタが FB3 の次のファイルエントリである FB4 を，LM の予備リンクポ

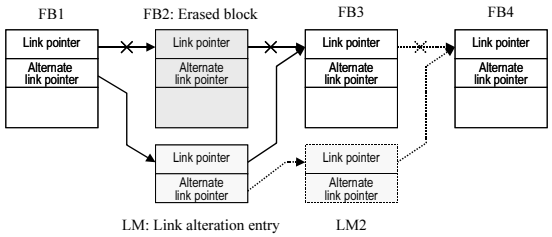


図 4 リンク書き換えエントリによるリンクの変更
Fig. 4 Modification of link using link alternation entry.

インタが LM2 を指すようにすればよい．

- ガベージコレクション
  実時間性を確保するために，1 回のガベージコレクションで，1 つの消去ブロックを処理することにした．また，ガベージコレクションの契機として，使用できる領域がまったくなくなったときとした．
  ガベージコレクションのアルゴリズムとして以下に示すように単純な方法を用いた．まったく使用されていないブロック（未使用ブロック）を用意しておき，そこにコピーすることによりガーベージコレクションを行う方法は，2.3 節で示す Microsoft Flash File System（MS-FFS）[7]でも採用されている．
  (1) ガベージコレクションのために，つねに 1 つ以上の未使用ブロックを用意しておく．
  (2) “現在の有効な使用領域+ブロック消去回数 × 2 KB” が最も小さくなるブロックを選択する．消去回数でバイアスしているのは，ブロック間の消去回数のばらつきを抑えるためである．
  (3) (2) で選択した消去ブロック内で，リンクが連続しているファイルエントリでグループ分けする．このとき，リンク書き換えエントリがリンクの途中に入っていても，連続しているものとする．
  たとえば，図 4 で FB1，FB2 および FB3 が選択した消去ブロックに含まれ，FB2 が削除されているものとする．この場合，FB1 → LM → FB3 → FB4 のリンクが存在するが，FB1 と FB3 が連続しているものとする．
  (4) グループ単位で，有効なファイルエントリのみを未使用ブロックにコピーする．このとき，途中にリンク書き換えエントリがあった場合は，回収される．
  (3) の例では，FB1，FB3 の順にコピーし，FB1 から FB3 に直接リンクして，LM を回収する．このとき，FB2 はコピーされないのでこの領域を回収できる．
  (5) グループの先頭を指しているファイルエントリの指す先を，コピー先のファイルエントリにするために，リンク書き換えエントリを用いて変更する．
  (6) コピー先のグループの最後のファイルエントリから次のファイルエントリを指すようにする．このとき，リンク書き換えエントリを介して次のファイルエントリを指していた場合は，直接リンクし，リンク書き換えエントリを回収する．
  (7) (4)〜(5) をグループがなくなるまで繰り返す．
  (8) (2) で選択した消去ブロックに対し，ブロック消去を行い，新たな未使用ブロックとする．

2.3 関連ファイルシステム

本節では，lffs と同様にフラッシュメモリを対象にした代表的なファイルシステムである，Flash Translation Layer（FTL）[6]および MS-FFS について説明する．

- FTL
  FTL は，インテルが PC Card 用に開発した磁気ディスクのエミュレータで，フラッシュメモリをファイルシステムから磁気ディスクとして見えるようにするものである．1 つの消去ブロックを複数のブロックに分割し，それを磁気ディスクのセクタに見せる．同じセクタに対する書き込みであっても，異なる消去ブロックに分散するように，セクタと消去ブロックを動的に対応させることにより，一部の消去ブロックにブロック消去が集中しないようにする．しかし，セクタとして割り当てるので，セクタの一部でも更新されると，セクタ全体の変更が生じる．
- MS-FFS
  マイクロソフト社が開発したバイト指向のファイルシステムで，lffs と同様に，リンク構造を用いたファイルシステムである．MS-FFS では，各消去ブロック内に固定のインデックス領域を設け，ファイルブロックの本体をインデックスを介して間接参照することにより，ガベージコレクションによるファイルブロックの移動に対処している．間接参照によって，ガベージコレクションが，lffs に比べ簡単になる．しかし，ファイル情報，デー

タ領域の情報，およびデータ領域が分離しており，それらの間で間接参照が行われるため，その分の時間的，空間的オーバヘッドが大きくなる．

3. lffs におけるメモリ利用効率の評価

文献 5) における lffs の評価では，ファイルの大きさの影響や同時書き込みの影響を考慮していなかった．本章では，ファイルの大きさ，1 回の書き込み操作での書き込み量および同時書き込み数をパラメータとして与え，その影響を調べた．

3.1 評価方法

評価は，文献 5) で作成したシミュレータおよび環境を用いて行った．消去ブロックの大きさ 64 KB，ブロック数 256 個のフラッシュメモリ（16 MB）において，以下の方法でファイルの作成，削除を行った．

(1) 指定された大きさのファイルを，指定された数だけ作成する．
(2) (1) で作成したファイルの 6%をランダムに削除する．ここで 6%という値は，実験で用いたパラメータでメモリフルを起こさない最大の値である．
(3) 削除したファイルと同数のファイルを指定された数で同時に作成する．このとき，1 回の書き込み操作での書き込み量を指定する．
(4) (2)，(3) を繰り返す．

1 組のパラメータにつき 5 回の実験を行い，その平均を求めた．

ファイルシステムが定常状態になるように，160 MB 分（対象のフラッシュメモリの 10 倍）のファイルを作成してから測定を開始し，480 MB 分（対象のフラッシュメモリの 30 倍）を作成するまでのログを得た．実験のパラメータとして，メモリの使用率，作成するファイルの大きさ，1 回に書き込む大きさ，同時に生成するファイル数を与えた．(1) で作成するファイル数は，与えられたファイルの大きさで，与えられたメモリの使用率になるようにした．また，このとき，管理情報を考慮に入れないでファイル数を決定した．このため，書き込み単位が小さくなると管理情報の影響が大きくなり，実際の使用率が高くなる．たとえば，書き込み単位が 512 B の場合，データの追記の管理情報が 16 B であるため，実際の使用率は与えられた使用率より約 3%高くなる．さらに，実際に作成するファイルの大きさは，与えられた大きさの ±20%の範囲で一様分散で与えた．

3.2 評価

表 1 のパラメータを用いて実験を行った．**図 5** に，同時に作成するファイル数が 1，5 の場合，および 1 回に書き込む大きさが 512 B と 16 KB の場合の実験結果を示す．図で，横軸はメモリの使用率を表し，縦軸はファイルの大きさに対する実際の書き込み量の比率を示す．たとえば，図 5 (c) で，メモリの使用率が 0.9 の場合，どのファイルの大きさに対してもファイルの大きさの約 7 倍の書き込みが発生していることになる．

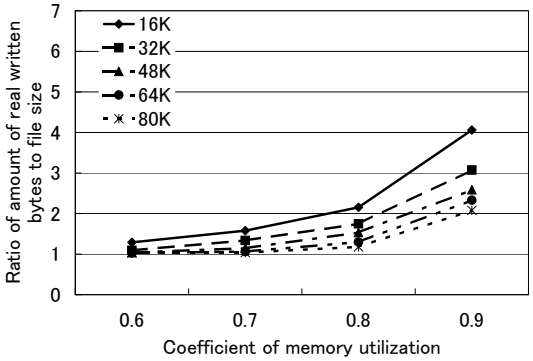
表 1 実験に用いたパラメータ
Table 1 Parameters of experiments.

Parameter	Value
Coefficient of memory utilization	0.6, 0.7, 0.8, 0.9
Average file size	16 KB, 32 KB, 48 KB, 64 KB, 80 KB
Write unit size	512 B, 1 KB, 2 KB, 4 KB, 8 KB, 16 KB
The number of concurrenct creation	1, 2, 3, 4, 5

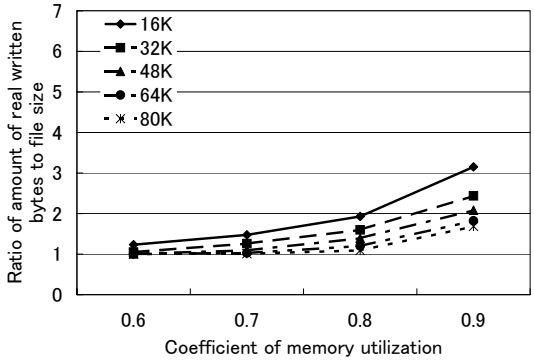
同時に書き込むファイル数に関しては，数が増えるほど差は小さくなるが，実際の書き込み量は同時に書き込むファイル数が多くなるほど増加した．また，1 回に書き込む大きさの他の場合に関しては，512 B と 16 KB 間で同じパラメータに対し，実際の書き込み量はログスケールでほぼ比例していた．これらをまとめると以下のようになる．

(1) メモリの使用率が低い場合は，実際の書き込み量はそれほど増加しない．また，書き込みの方法およびファイルの大きさにほとんど影響されない．
(2) メモリの使用率が高くなるほど，実際の書き込み量が増加し，書き込みの方法およびファイルの大きさに大きく影響される．メモリの使用率が 1 に近づくと，実際の書き込み量が急激に増加する．
(3) 同時に作成するファイル数が 1 個の場合，実際の書き込み量は，1 回に書き込む大きさにそれほど影響されない．
(4) 同時に作成するファイル数が 1 個の場合，ファイルの大きさが大きいほど，実際の書き込み量が減少する．
(5) 同時に書き込むファイル数が増加するほど，実際の書き込み量が増加する．
(6) 同時にファイルを作成する場合，1 回に書き込む大きさが小さいほど，実際の書き込み量が増加する．

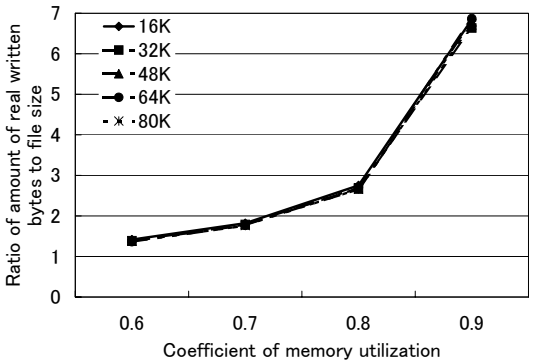
メモリの使用率が高くなるにつれて，各消去ブロックで実際に使用されている領域が増加する．そ



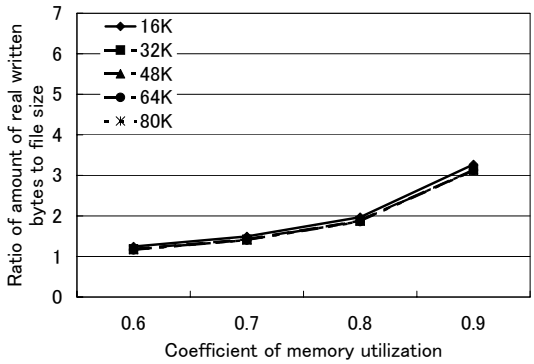
(a) The number of concurrent creation: 1, Write unit size: 512 B



(b) The number of concurrent creation: 1, Write unit size: 16 KB



(c) The number of concurrent creation: 5, Write unit size: 512 B



(d) The number of concurrent creation: 5, Write unit size: 16 KB

図 5 実験結果
Fig. 5 Results of experiments.

のため，ガベージコレクションで書き戻す領域が増加するとともに，回収できるメモリの領域は小さくなる．すべての消去ブロックの有効な使用領域が一様であると仮定すると，使用率が α なら，ガベージコレクションで書き戻すメモリ量（S_{WB}）は $\alpha \times$ 消去ブロックの大きさ となり，回収される領域（S_{RC}）は $(1-\alpha) \times$ 消去ブロックの大きさ となる．これは，S_{RC} の領域を書き込むために，$S_{RC}+S_{WB}$ の書き込みが必要であることを示す．したがって，実際の書き込み量は $(S_{RC}+S_{WB})/S_{RC} = 1/(1-\alpha)$ 倍になる．各消去ブロックでの有効な使用領域にばらつきがあるため，これ以上の値になることはない．特に，1 つのファイルが 1 つの消去ブロックに配置されるような場合は，1 つのファイルを削除することによって，そのファイルが占めていた領域がすべて回収できる領域になり，実際の書き込み量を少なくする．この影響は，メモリの使用率が上がるほど大きくなる．これらの理由により，(1)，(2) で述べたようになる．

単独にファイルを作成する場合は，**図 6** の左上に示すように，メモリの空き領域に 1 つのファイルのファイルエントリが連続して配置され，少数の消去ブロックに配置される割合が高くなる．このため，図の右上に示すように，1 つのファイルを削除したときに，ガベージコレクションで回収できる領域の大きい消去ブロックが発生する．したがって，単独にファイルを作成する場合の実際の書き込み量は，$1/(1-\alpha)$ よりはるかに小さくなる．1 回に書き込む大きさが変化して

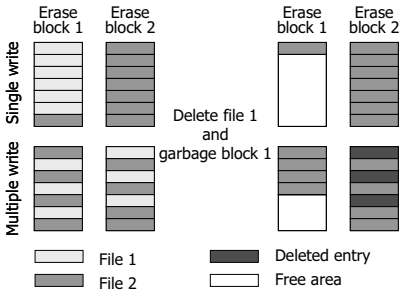
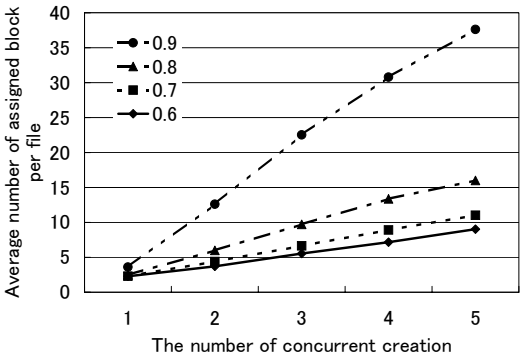
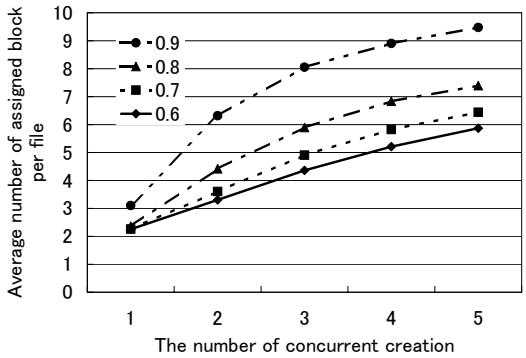


図 6 同時作成の影響
Fig. 6 Influence of multiple creation.



(a) File size: 80 KB, Write unit size: 512 B



(b) File size: 80 KB, Write unit size: 16 KB

図 7 同時書き込み数の影響
Fig. 7 The influence of concurrent creation.

も，管理領域の数が変化するだけで，連続して配置されることには変わりがないので，(3) で述べたように，実際の書き込み量は 1 回に書き込む大きさにそれほど影響されない．また，ファイルの大きさが大きくなると，1 つのファイルを削除するときに回収できる領域が大きくなるので，(4) で述べたようなる．

これに対して，複数のファイルを同時に作成する場合は，図 6 の左下の部分に示すように，複数のファイルのファイルエントリが交互に配置される．ガベージコレクションの契機をメモリの空き領域がなくなったときとしたことにより，1 つの消去ブロックからメモリが割り当てられるようになったためである．これにより，1 つのファイルが複数の消去ブロックに分散して配置されることが多くなる．このため，図の右下に示すように，1 つのファイルが消去されても，回収できる領域が，複数の消去ブロックに分散し，1 つの消去ブロックで回収できる領域が，単独にファイルを作成した場合と比較して小さくなる．結果として，実際の書き込み量は，単独にファイルを作成する場合より多くなる．

同時に作成するファイルの数が増えると，より多くのファイルが 1 つの消去ブロックを共有するようになるため，1 つの消去ブロックに占める領域が小さくなり，より多くの消去ブロックに分散する．このため，(5) で述べたように，同時に書き込むファイル数が増加するほど，実際の書き込み量が増加する．このことを確認するために，1 つのファイルのファイルエントリが含まれる消去ブロックの数を測定した．**図 7** は，ファイルの大きさが 80 KB の場合における結果を示す．横軸は同時書き込み数，縦軸は 1 つのファイルのファイルエントリが含まれる消去ブロックの平均数を表す．1 つのファイルのファイルエントリが含まれる消去ブロックの数は，1 回の書き込み量やメモリの使用率に影響され，同時書き込み数が多くなるほど増加している．特に，同時書き込み数が少ない間や，1 回の書き込み量が少ない場合は，同時書き込み数に比例して増加している．

また，1 回の書き込み量が小さくなると，図 7 に示されるように 1 つのファイルのファイルエントリが含まれる消去ブロックの数が多くなり，より一様に分散し回収できる領域もより平均化されるため，突出して回収できる消去ブロックが少なくなる．このため，(6) で述べたように，1 回に書き込む大きさが小さいほど，実際の書き込み量が増加する．図 5 の (c)，(d) において，すべてのファイルの大きさでほとんど同じ結果が出ているのは，どちらの場合においても，1 回あたりの書き込み量が最小のファイルの大きさ（16 KB）以下であるため，どのファイルの大きさでも，1 つの消去ブロックに占める大きさに差が出なかったためである．

4. lffs のメモリ割当てアルゴリズムの改良

3 章で述べたように，複数のファイルを同時に作成するときに，1 つの消去ブロックからそれらのファイルにメモリを割り当てることにより，1 つのファイルが多くの消去ブロックに分散して配置される．これを避けるために，lffs のメモリ割当てアルゴリズムを改良する．

ファイルを作成するときに，それぞれのファイルに空き領域を持つ消去ブロックを独立して割り当て，割り当てられた消去ブロックからメモリを取り出すことにより，1 つのファイルが少数の消去ブロックに配置されるようになる．改良前の lffs でも，この方針でメモリを割り当てるようにしていたが，ガベージコレク

ションの契機を使用できるメモリがまったくなくなったときとしたため，メモリをすべて使い切った後は，つねに 1 つの消去ブロックにのみ空き領域が残るようになり，個々のファイルに，独立した消去ブロックを割り当てることができなくなっていた．

そこで，以下の手順で消去ブロックをファイルに割り当てることにした．

(1) ファイルを新規作成するとき
他のファイルに割り当てられていない，空き領域を持つ消去ブロックがある場合は，それらの中から，最も空き領域が大きいものを割り当てる．
ない場合は，ガベージコレクションにより，空き領域を持つ消去ブロックを生成して割り当てる．

(2) ファイルを書き込みモードで開くとき
開いたファイルの最後のエントリが含まれる消去ブロックに空き領域がある場合，そのブロックを割り当てる．
ない場合は，(1) と同じ方法で，空き領域を持つ消去ブロックを割り当てる．

(3) 割り当てられた消去ブロックの空き領域を使い切ったとき
(1) と同じ方法で，空き領域を持つ消去ブロックを割り当てる．

さらに，すでにファイルに割り当てられている，空き領域を持つ消去ブロックがガベージコレクションの対象にならないように，ガベージコレクションの 2 番目のステップで，これらのブロックを選択の対象外となるように変更した．

これにより，それぞれのファイルに独立して消去ブロックが割り当てられるので，それぞれのファイルを単独に作成したときと同様の，メモリ割当てが行われるようになる．

また，ファイルが多くの消去ブロックに分散しないように，ガベージコレクションの対象として，あるしきい値以上の空き領域を持つ消去ブロックがなく，かつあるしきい値以上の空き領域を確保できる消去ブロックがあれば，そのブロックをガベージコレクションして割り当てる方式も考え，実装してその効果を調べた．しかし，しきい値をどのように変えても，すべての領域を使い切ってからガベージコレクションを行う方式の方が，良い結果が得られた．この理由は，空き領域を使い切るまで待つことによって，削除されるファイルが発生し，それにより，回収できる領域が増加するためである．つまり，空き領域を使い切ったために，より多くの消去ブロックに分散されるファイルが発生することによる性能低下よりも，削除されるファイルが発生することによる性能向上の方が大きかったことになる．

5. 改良した lffs の評価

4 章で述べた方法を組み込んだ lffs を 3 章とまったく同じパラメータで評価した．その結果を**図 8** に示す．図 8 の (a)，(b)，(c)，(d) は，それぞれ図 5 の (a)，(b)，(c)，(d) で使用したパラメータと同じものを用いている．

まず，単独にファイルを作成する場合について評価する．図 5 (a) と図 8 (a) および図 5 (b) と図 8 (b) どうしを比較すると，全体の傾向はまったく同じであるが，メモリの使用率が 0.9 の場合，わずかに改良後の方が良くなっている．しかし，シミュレーションの精度を考慮すると，誤差範囲とも考えられ，ファイルへの消去ブロックの割当てアルゴリズムの改良が影響したかは判断できない．

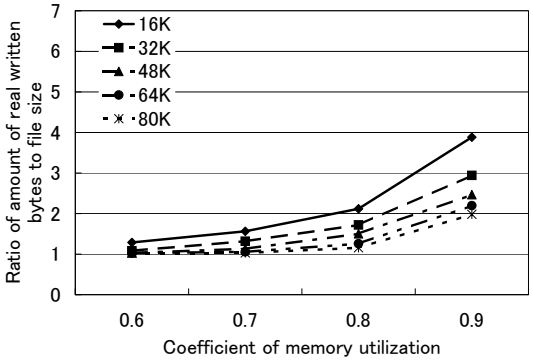
次に，複数のファイルを同時に作成する場合について評価する．図 5 (c) と図 8 (c) および図 5 (d) と図 8 (d) どうしを比較すると，1 回に書き込む大きさが 512 B でも 16 KB のどちらの場合でも改善されている．特に，1 回に書き込む大きさが 512 B でメモリ使用率が 0.9 の場合，実際の書き込み量が，ファイルの大きさが小さい場合（16 KB）で 0.56 倍，大きい場合（80 KB）で 0.26 倍になっており，改良の効果が大きく現れている．

最後に同時書き込み数の影響を調べる．図 8 (a) と図 8 (c) は同時書き込み数以外は同じパラメータである．まったくといってよいほど同じ結果を示している．また，1 回に書き込む大きさが 512 B から 16 KB に変わっても，図 8 (b) と図 8 (d) で示すように，同時書き込み数の影響を受けてない．この結果は，**図 9** で検証できる．図 9 は図 7 と対応し，同時書き込み数に対する 1 つのファイルのファイルエントリが含まれる消去ブロックの平均数を示している．改良前は，1 つのファイルのファイルエントリが含まれている消去ブロックの平均数が同時書き込み数にほぼ比例して増加していたが，改良後は，同時書き込み数にほとんど影響されず，ほぼ一定である．

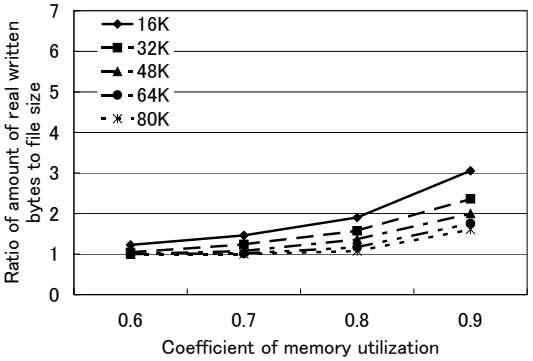
以上の結果より，ファイルへの消去ブロックの割当てのアルゴリズムの改良の効果が現れているのが確認できた．

6. おわりに

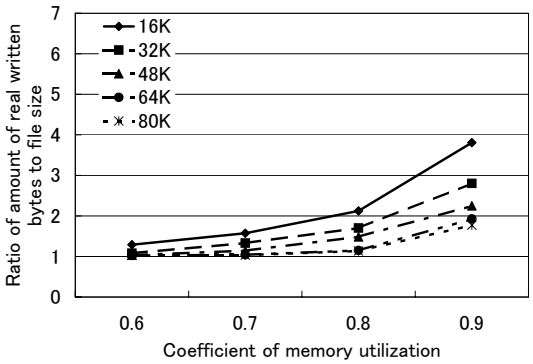
本論文では，フラッシュメモリのためのリンク構造



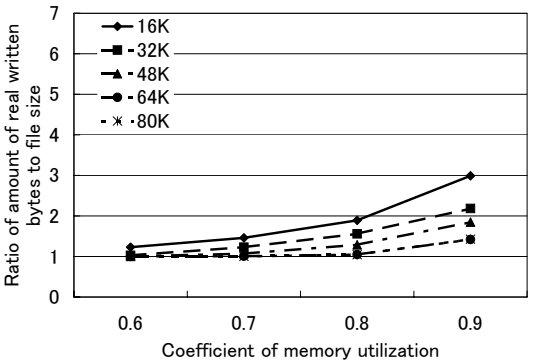
(a) The number of concurrent creation: 1, Write unit size: 512 B



(b) The number of concurrent creation: 1, Write unit size: 16 KB

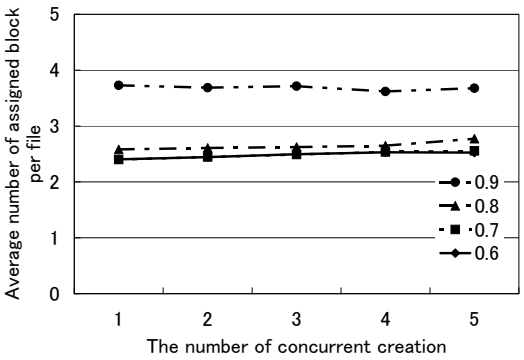


(c) The number of concurrent creation: 5, Write unit size: 512 B

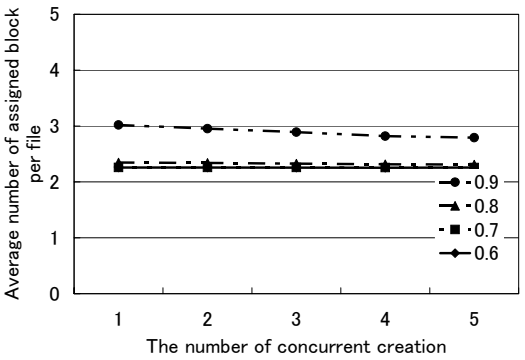


(d) The number of concurrent creation: 5, Write unit size: 16 KB

図 8 改良した lffs での実験結果
Fig. 8 Results of experiments on new lffs.



(a) File size: 80 KB, Write unit size: 512 B



(b) File size: 80 KB, Write unit size: 16 KB

図 9 改良した lffs における同時書込み数の影響
Fig. 9 The influence of concurrent creation on new lffs.

ファイルシステム lffs におけるメモリ割当ての効果について述べた．旧バージョンの lffs では，ガベージコレクションの契機を，使用できるメモリがなくなったときとしていたため，複数のファイルを同時に作成する場合に，実際に書き込むメモリ量が単独にファイルを作成する場合より多くなっていた．特に，メモリの使用率が高くなるにつれ，その差が大きくなっていた．この問題を解決するために，作成するファイルに異なる消去ブロックを割り当てるようにした．この結果，複数のファイルを同時に作成する場合に実際に書き込

むメモリ量が，単独にファイルを作成する場合とほとんど変わらなくなった．

この結果は，いい換えれば，消去の対象となるもの（ファイルシステムではファイルであるが）をより少ない消去ブロックに集めることができれば，ガベージコレクションにより，より多くのメモリを回収できる，ということになる．この結果は，フラッシュメモリを対象としたシステムで，ガベージコレクションを必要とするシステムに対して適用できるものと考えている．

今後は，実際のアプリケーションにおけるファイルシステムの想定した，あるいはファイル操作のログを用いた，より実践的な評価が必要である．

参 考 文 献

1) 日経 BP 社：大規模フラッシュ・メモリー—音楽用の記録媒体の主役へ前進，pp.24–25，日経マイクロデバイス (1998).
2) BYTE-WIDE SmartVoltage FlashFile™ MEMORY FAMILY 4, 8 AND 16 MBIT, Intel (1997).
3) Brown, W.D. and Brewer, J.E.: Nonvolatile Semiconductor Memory Technology, The institute of Electrical and Electronics Engineers, Inc. (1997).
4) 甲斐，井上，安浦：フラッシュメモリを主記憶とするシステムのためのメモリ・アーキテクチャの検討，電子情報通信学会技術研究報告，CPSY98-111, pp.51–58 (1998).
5) 石墨，最所，福田：組み込みシステム向けフラッシュメモリファイルシステムの設計，電子情報通信学会論文誌，Vol.J84-D-I，No.1，pp.90–99 (2001).
6) Understanding the Flash Translation Layer (FTL) Specification, Intel (1997).
7) 田中康之：フラッシュファイルシステムのファイル管理技術，インタフェース，Vol.26，No.1，pp.176–184 (2000).

(平成 12 年 12 月 19 日受付)
(平成 13 年 4 月 6 日採録)

最所 圭三（正会員）

1959 年生．1982 年九州大学工学部情報工学科卒業．1984 年同大学院工学研究科修士課程修了．同年同大学工学部助手．1991 年同大学工学部講師．1993 年同大学大型計算機センター助教授．1994 年奈良先端科学技術大学院大学情報科学研究科助教授．2000 年香川大学工学部教授，現在に至る．工学博士．高信頼性システム，並列/分散処理，モバイルシステム，並行処理等の研究に従事．1998 年情報処理学会全国大会大会優秀賞受賞．IEEE Computer Society，電子情報通信学会，日本ソフトウェア科学会各会員．

福田 晃（正会員）

1954 年生．1977 年九州大学工学部情報工学科卒業．1979 年同大学院工学研究科修士課程修了．1979 年同大学院修士課程修了．同年 NTT 研究所入所．1983 年九州大学大学院総合理工学研究科助手．1989 年同大学助教授．1994 年奈良先端科学技術大学院大学情報科学研究科教授．2001 年より九州大学大学院システム情報科学研究院教授．工学博士．オペレーティング・システム，組込システム，並列化コンパイラ，計算機アーキテクチャ，並列/分散処理，性能評価等の研究に従事．本学会平成 2 年度研究賞，平成 5 年度 Best Author 賞受賞．著書「並列オペレーティングシステム」（コロナ社），訳書「オペレーティングシステムの概念」（共訳，培風館）．ACM，IEEE Computer Society，電子情報通信学会，日本 OR 学会各会員．