組み込み向けごみ集めにおける退去区画の選択

1211003　赤澤 亮弥　　岩崎・中野研究室

1　背景

近年，モバイル端末向けのプラットフォームとして，Google が中心となって開発している Android が普及している．

Android アプリケーションは Java で記述される．Java はオブジェクト指向言語であり，実行時にオブジェクトと呼ばれるデータを大量に作る．このとき，オブジェクトはヒープと呼ばれる領域から割り当てられる．Java にはごみ集め（Garbage Collection，以下 GC）と呼ばれる機構が備わっており，実行時，ヒープ中の不要になったオブジェクトを自動的に回収する．これにより，使用中のオブジェクトを誤って解放してしまったり，逆に不要になったオブジェクトを解放し忘れるなど，プログラマが手動でオブジェクトを解放しようとして発生しうる諸問題を回避している．

2　GC

GC において，あるオブジェクトが不要であるかどうかの判断は，そのオブジェクトが今後参照される可能性の有無によって行われる．参照される可能性のあるオブジェクトは「生きている」とし，参照される可能性がないオブジェクトは「ごみ」として回収される．オブジェクトの参照可能性を調べるには，ルート集合から参照を辿ればよく，それによって到達したオブジェクトは「生きている」オブジェクトであり，到達できなかったオブジェクトは「ごみ」であると判別する．ここで 4 ルート集合とは，プログラムが直接操作できる領域のことで，スタック，レジスタ，グローバル変数などが該当する．

Android で主に使用されている GC のアルゴリズムは，Mostly Concurrent GC という並行マークスイープ GC である．マークスイープ GC は，ルート集合から到達できる全てのオブジェクトにマークするマークフェーズと，マークされなかったオブジェクトを「ごみ」として回収するスイープフェーズによって構成される．単なるマークスイープ GC ではこれを，アプリケーションの処理（以下，ミューテータと呼ぶ）を停止した状態で実行するが，Mostly Concurrent GC ではこれを，出来る限りミューテータを動作させた状態で実行する．

3　断片化

オブジェクトの割り当てと「ごみ」の回収を繰り返していくと，図 1 のように空き領域が不連続的に生まれ，総量としては十分にもかかわらず新たなオブジェクトを割り当てることができないということが起こり得る．このように不連続的な細かい空き領域が生まれることを断片化と呼ぶ．

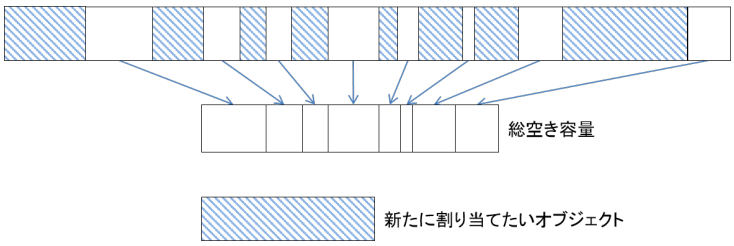

図 1　断片化

4　コンパクション

断片化を解消するには，図 2 のようにオブジェクトを移動させて空き領域を連続させればよく，これをコンパクションと呼ぶ．ただし，図 2 のように全てのオブジェクトを移動させるというのは処理時間のコストが大きい．よって，断片化の解消と実時間性の両立を考えると，より部分的なコンパクションの方法を考える必要がある．

5　目的と方針

本研究では，断片化の解消と実時間性を両立させる部分的なコンパクションを Android 上に実装することを目的とする．

そのための方針として，部分的なコンパクションとして複製コンパクションを採用し，Android4.4 以降のランタイムである ART に対して実装を行う．また複製コンパクションの実装に際し，退去区画の選び方が複数考えられ，それが GC の性能に繋がるため，より効率の良い退去区画の選び方を比較検討する．

6　本研究における GC

本研究の GC は，マークフェーズとコンパクションフェーズで構成される．マークフェーズで「生きている」オブジェクトをマークした後（図 3(a)），コンパクション

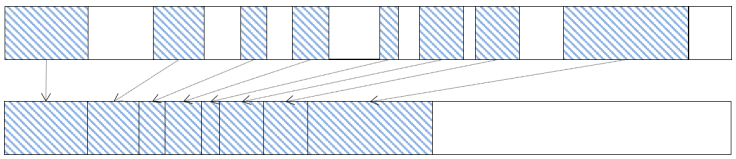

図 2　コンパクション

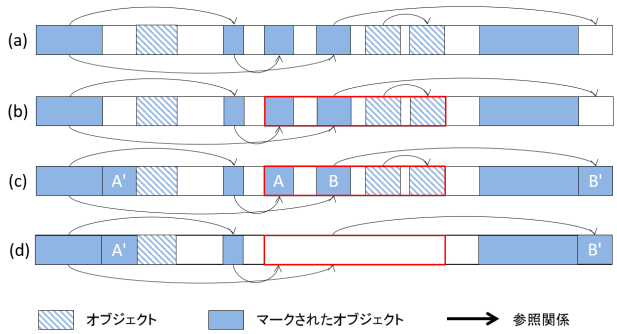

図 3　複製コンパクション

フェーズに入るが，このとき複製コンパクションを利用する．複製コンパクションではまず，ある程度の大きさの領域をヒープ上に設定する．これを退去区画と呼ぶ（図 3(b)）．そして退去区画内の「生きている」オブジェクトを区画外の空いている領域にコピーし（図 3(c)），その後退去区画内のオブジェクトを全て「ごみ」として回収する（図 3(d)）．

7　関連研究

関連研究として退去区画の選び方に関する研究があり [1]，この研究では最も断片化の酷い領域を計算し，退去区画として設定している．

8　現状と今後

現状では Android OS のビルド及びエミュレートの環境構築が終了し，今後は ART に実装していく．具体的には，初めにミューテータを止めた状態で本研究の GC を行うように実装し，動作確認を行う．その後，退去区画をどの場所にどの大きさで設定するべきかを検討し，最後に上記の実装を拡張してミューテータを動かした状態で動作するようにし，ベンチマークプログラムによる性能評価を行う．

参考文献

[1] Tomoharu Ugawa, Hideya Iwasaki, and Taiichi Yuasa. Improved replication-based incremental garbage collection for embdded systems. In *Proceedings of the 2010 international symposium on memory management - ISMM '10*, pages 73-82, 2010.

[2] 鵜川始陽, 信岡孝佳, 海野弘成, 湯淺太一. 書込みバリアにロックやメモリバリア命令を用いない並行スナップショットごみ集め. **コンピュータソフトウェア**, 29(3):143-156, 2012.

[3] 中野陽基 (2015), "Android 上のごみ集めにおける停止時間の削減", 電気通信大学大学院情報理工学研究科修士論文