Vol.2010-OS-113 No.9
2010/1/28

省メモリのためのメモリアクセス解析手法の提案と実装

壬生 亮太[†1] 高橋 雅彦[†1] 菅原 智義[†1]

携帯電話等の組み込み分野ではメモリ不足が問題となっている．メモリ使用量を削減するため，複雑化したアプリケーションのメモリ使用状況を解析する必要がある．エミュレータ等によりメモリアクセスのログを取得することはできるものの，性能劣化や膨大なログが問題である．このためメモリアクセスの状況把握からメモリ使用量の削減が可能な箇所を特定することが困難になっている．本稿では，省メモリを実現するためのメモリアクセス解析手法として，ストリーミング可視化と使用ページに絞った可視化を提案する．ストリーミング可視化によりメモリアクセス全体を俯瞰的に把握し，使用ページに絞った可視化により改善可能な箇所の特定を行う．ストリーミング可視化のために共有メモリを用いた性能改善を行った．これを評価した結果，エミュレーションのみの場合に比べて wc のメモリアクセスログの取得によるオーバヘッドが 7%に抑えられた．

Proposal and Implementaion of Memory Access Analysis Method for Memory Saving

Ryota Mibu,[†1] Masahiko Takahashi[†1]
and Tomoyoshi Sugawara[†1]

Although lack of memory has been problem in embedded systems such as mobile phones, it is hard to understand application's memory use due to its complexity. To analyse memory use, a memory access log is acquirable by a system emulater, with enormas log and deterioration in performance. In this paper, we propose an intuitive method of memory access analysis without losing accuracy, by streaming visualization and its optimized view. After implementing our proposed method on an emurator ("valgrind"), we found that the overhead in emulating and acquiring memory access of "wc" (word counter) is only 7% compared to nullgrind.

1. はじめに

携帯電話等の組み込み分野ではメモリ不足が深刻な問題となっている．それでもなお新規のアプリケーションの導入が期待されており，アプリケーションのメモリ使用量を削減することが求められている．ところが，アプリケーションは複雑化しそのメモリ使用状況の把握が困難になっている．プロファイラやエミュレータ等により，メモリ使用量やメモリアクセスのログが取得できるものの，それらの情報では省メモリにつながりにくく，メモリ使用量だけでは改善箇所を特定できない．また，メモリアクセスのログは膨大であり，人が見て把握できるものではない．しかしながら，このメモリアクセスログをうまく解析できれば，ライブラリやプログラムのメモリ使用量を削減できる箇所を特定できる．例えば，起動時に読み込まれるだけで使われない関数や変数を特定し削減すること，あるいは 1 度しか使わないデータがあれば使われた後に他のデータを置いてメモリ使用量を減らすことが考えられる．メモリ使用量や関数／変数の使用回数のプロファイルなどのように時間やアドレスといった情報を省いたものでは，このような解析は得られない．

本稿では，エミュレータにより取得した膨大なメモリアクセスログを時間やアドレスといった情報を省かずその全体を把握するため，実行命令順とアドレスを軸にメモリアクセスを可視化する 2 つの手法を提案する．メモリアクセス全体を俯瞰し，直感的に理解できるストリーミング可視化方法と，バイト単位の調査を行える使用ページに絞った可視化方法である．省メモリにつながるメモリ使用状況の把握が目的であるため，これらの可視化によりメモリ領域からバイト単位のアクセスを把握し，ページ単位の削減を行う．

エミュレータによるログ取得では，エミュレーションオーバヘッドとログ書き出しでの性能劣化が問題となっている．我々が提案するストリーミング可視化では処理性能を改善することで，エミュレートしたプログラムの GUI を操作しながらより直感的なメモリアクセスを把握できる．エミュレータのログ書き出しに共有メモリを用いることでエミュレーションのみの場合と比べ処理速度の低下を 1.1〜2.2 倍に抑えた．

2. 省メモリのためのメモリアクセス解析

組み込みシステムの開発において省メモリを実現するためには，メモリアクセスの解析が

†1 NEC システムプラットフォーム研究所
System Platforms Research Laboratories, NEC Corporation

Vol.2010-OS-113 No.9
2010/1/28

```
I  0023C790,2
I  0023C792,5
 S BE80199C,4
I  0025242B,3
 L BE801950,4
I  0023D476,7
 M 0025747C,1
I  00252305,1
 L 00254AEB,1
 S 00257998,1
```

図 1 valgrind 付属の解析ツール lackey によるメモリアクセスログの出力.

必要である．メモリアクセスログはエミュレータを用いることで取得できる．このとき，メモリアクセスログが膨大であるという問題やエミュレーションとログを書き出すことで性能劣化するという問題がある．また，それらの膨大なログを人が見ても直感的に理解できない．メモリアクセスを可視化することで直感的な理解を図ることができるが，どうメモリアクセスの可視化を行うかが問題となっている．本節では「膨大なメモリアクセスログ」，「エミュレーションとログ書き出しによる性能劣化」及び「膨大なメモリアドレス空間でのメモリアクセス可視化」を課題として述べる．

2.1 膨大なメモリアクセスログ

エミュレーションオーバヘッドの小さい実行基盤として valgrind[1] が広く用いられている．valgrind はエミュレーションにより動的な解析を行うフレームワークであり，エミュレータと複数の解析ツールで構成されている．解析ツールには様々なデバッガやプロファイラが用意されているおり，新たなツールも追加できる．解析ツールの例として用意されている lackey ではメモリへのアクセスのログを取得でき，**図 1** のようにメモリアクセスの種類，アドレス，バイト数を出力する．メモリアクセスの種類はアルファベットで表され，“I” は Instruction，“S” は Store，“L” は Load，“M” は Modify を意味する．Modify とはインクリメント命令により同じアドレスに対して Load，Store が行われたことを意味する．

このようなプリミティブなメモリアクセス情報ではそのログサイズが膨大になる．このためアプリケーション開発者は改善が必要な箇所を特定することが困難になっている．QEMU[2] を使い 5〜10 秒かかる OS 起動処理について物理メモリアドレスへのアクセスログを取得した際，1.7GB〜10GB のログが出力されることが報告されている[3]．このような理由からメモリアクセスログの取得だけでは省メモリにつながりにくい．

2.2 エミュレータとログ書き出しによる性能劣化

エミュレータを使ったメモリアクセスログの取得では，エミュレーションによる性能劣化が問題となる．valgrind ではエミュレーションオーバヘッドにより 3〜5 倍ほど実行速度が低下する[4]．また，メモリアクセスのログをテキスト形式で書き出すことが問題となっている．QEMU を使い物理メモリアドレスへのアクセスログをテキスト形式で書き出した際，エミュレーションのみの場合に比べて 10 倍〜23 倍の速度低下が報告されている[3]．

2.3 膨大なメモリアドレス空間でのメモリアクセス可視化

メモリアクセスを把握するために様々な可視化方法がある．それらは時間を軸にするもの，アドレスを軸にするもの，そして時間とアドレスを軸にするものに分類される．時間を軸に可視化する方法は，プログラム実行全体からどこでキャッシュミスやページフォルト等のイベントが発生したかを特定するためのものである[5]．また，アドレスを軸に可視化する方法はメモリアドレスがどのように使われているかを把握することが目的である．例えば，キャッシュを含めどのメモリへのアクセスが発生したかをアニメーションで可視化する手法がある[6]．これは時系列全体を俯瞰してみることはできないが，プログラムの特定部分でのデータの流れを細かくモニタリングできる．そして，時間とアドレスを軸に可視化する方法はメモリアクセスを俯瞰し直感的な理解を得るために行うもので，メモリアクセスパターンを把握する場合やプログラムの評価に用いる場合がある[7]．

省メモリを実現するためにメモリアクセスを可視化する場合，32 ビット OS で 4GB ある仮想アドレス空間すべてをどう表現するかが問題となる．メモリアクセスについて全体の状況把握からメモリ使用量を削減できる箇所を特定するためには，メモリアクセスを俯瞰でき，かつバイト単位で調査できることが必要である．メモリアクセスの時間やアドレスといった情報を省かずとも，膨大なメモリアクセスを直感的に把握できなければならない．

3. 解決のためのアプローチ

我々は省メモリを実現するために実行命令順とアドレスを軸にメモリアクセスを可視化する解析手法を提案する．我々の提案するメモリアクセスの解析手法では，全体を俯瞰するスケールとバイト単位で分析できるスケールに対応すべく 2 つの可視化方法がある．それら

は「ストリーミング可視化方法」と「使用ページに絞った可視化方法」である．本節では提案するこれらの可視化方法とそれらを実現する機能の構成を説明し，それらによって可能になるメモリ使用量の削減例について述べる．

3.1 実行命令順とアドレスを軸にしたメモリアクセスの可視化

メモリアクセスを実行命令順とアドレスを軸に 2 次元で可視化する．エミュレーションによりメモリアクセスを取得するため正確な時間が取得できない．このため実行命令順を軸とした．どちらの軸も膨大な範囲になるため，直感的に理解できるよう全体を俯瞰するスケールから使用アドレスの特定や使用バイト数を把握できるスケールまでを考える．このため，我々が提案する手法ではメモリアクセスをストリーミング可視化方法と使用ページに絞った可視化方法の 2 つを用いる．メモリアクセスの解析はストリーミング可視化方法で見当を付け，使用ページに絞った可視化方法で詳しく調査するという流れで行う．これによりメモリアクセス全体を素早く直感的に把握した上で特定領域を詳細に分析できる．

3.1.1 ストリーミング可視化方法

ストリーミング可視化方法ではエミュレータにより出力されたメモリアクセスログを直ちに可視化する．すべてのログを保存する必要がなくなり，ログファイルが巨大化する問題が解消される．また，エミュレーション対象のプログラムを GUI で操作し，その画面表示を確認しながらメモリアクセスを確認でき，より直感的な把握が可能になる．

3.1.2 使用ページに絞った可視化方法

もう 1 つの使用ページに絞った可視化方法は分析対象となるアドレス領域のみを可視化する．バイト単位で取得したアクセスログをすべて保持しておき，プログラムの実行を通して使用されなかったページを表示しないことでアドレス軸の描画スケールを最適化する．ストリーミング可視化と異なり，何度もエミュレーションを実行するものでなく，特定のアドレス領域におけるメモリアクセスログを収集し，より詳細な分析を行うためのものである．

3.2 構　　成

提案する可視化方法を実現する機構はエミュレータと可視化ツールで構成され，エミュレータでメモリアクセスログを取得し，可視化ツールで描画する．

エミュレータからストリーミング可視化ツールへデータを渡す方法は，共有メモリを使う方法（**図 2**）とソケット通信を用いる方法（**図 3**）がある．共有メモリを使う方法ではリングバッファを用いてメモリアクセスログを渡す．共有メモリを用いることでログ書き出し処理が高速化される．ソケット通信を用いる方法ではプログラムを実行するマシンとモニタリングするマシンを分けることでハードウェアの制約を避けられる．この 2 つのデータを渡す方法でログボリュームの制限が無くなる．また，オーバヘッドの改善ためバイナリ形式でメモリアクセスのログを扱う．このように処理性能の低下をできる限り抑えることで，エミュレーション対象のアプリケーションの GUI 操作や画面表示と合わせながらメモリアクセスを把握できる．

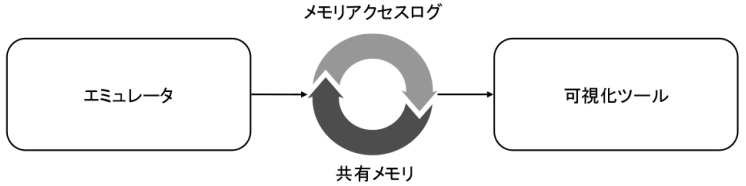

図 2　共有メモリを使い同一マシン上でエミュレートと可視化を行う．

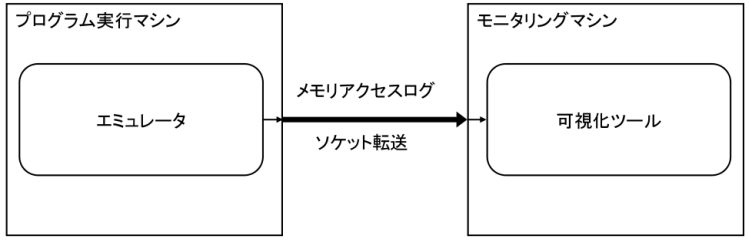

図 3　ソケット通信により異なるマシン上でエミュレートと可視化を行う．

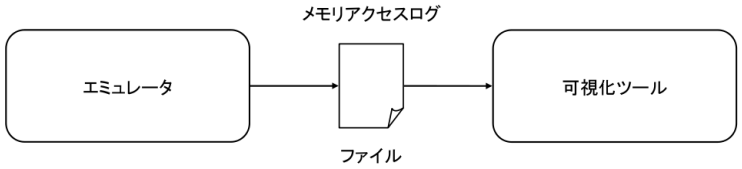

図 4　メモリアクセスログをファイルに書き出し，可視化を行う．

さらに，使用ページに絞った可視化方法ではプログラムの実行全体のログが必要となるため，**図 4** のようなファイルへのログ書き出し機能も持つ．また，使用ページに絞った可視化方法では特定のアドレス領域を調査するため，必要なアドレス領域やメモリアクセスの種類に絞ったログの取得を可能にする．

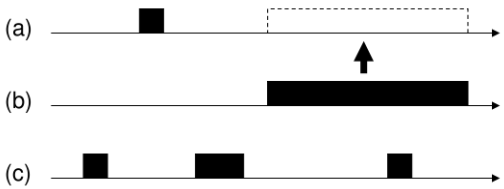

図 5 実行命令順に各ページで発生したメモリアクセス

3.3 メモリ使用量の削減例

我々の提案する解析手法により，実行命令順あるいはアドレス軸でのみ可視化した場合では見つけられなかったメモリ使用量の削減可能な箇所を新たに発見できる．例えば，メモリ管理にはページング方式が採用されており，実行したいプログラムが汎用的な共有ライブラリを読み込む際，必要でない関数も読み込まれることがあり，これらを特定し，共有ライブラリの関数を並び替えることで，実行時のメモリ使用量を削減できる．また，プログラムの実行を通して特定の間だけ使われるページを重ねることでメモリ使用量を削減できる．これは**図 5** を用いて説明する．プログラムの実行を通して図 5 のようなメモリアクセスが発生した場合，ページ (a) は初めてページ (b) がアクセスされた以降にアクセスされなくなる．この場合，ページ (a) をページ (b) として使うことで 1 ページ分のメモリ使用量が削減できる．ただし，同じようにページ (a) とページ (c) を一緒にすることはできない．ページ (c) はページ (a) が使用される前後でアクセスされており，保存していたデータを適切に参照できないからである．

4. 設　　計

本節では，共有メモリによるエミュレータから可視化ツールのデータ渡し，及び可視化ツールの設計について述べる．

4.1 共有メモリによるデータ渡しの設計

ストリーミング可視化では共有メモリでデータを渡すため，リングバッファを用いる．エミュレータはアクセスしたメモリアドレスに付加情報を加えて共有メモリ上のリングバッファにバイナリ形式でログを書き出す．付加する情報は，メモリアクセスの種類とアクセスしたバイト数である．ログ書き出しのオーバヘッドを抑えるため，書き出すログは特定のものを指定できるようにすることが重要である．このため，取得するメモリアクセスログは特定のアドレス範囲と特定のメモリアクセスの種類に限定出きるようにする．可視化ツールはリングバッファに書き込まれたメモリアクセスログを取得する．

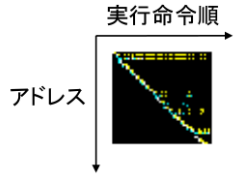

図 6 実行命令順とアドレスを軸にしたメモリアクセスの可視化

4.2 可視化ツールの設計

可視化ツールは受け取ったメモリアクセスログを**図 6** のように，横軸を実行命令順，縦軸をアドレスとした 2 次元の領域にビットマップ形式で描画する．メモリアクセスの種類は色で表す．

4GB の仮想アドレスをバイト単位で PC 画面に描画することは現実的に不可能であり，人が見ても直感的にメモリアクセスを把握できるものにならない．そのため，実行命令順とアドレスの両方の軸について 1 ピクセルに複数のアドレスやメモリアクセスをまとめる．1 ピクセルの表すアドレス範囲で複数のメモリアクセスがあるため，メモリアクセスの種類とそれらの組み合わせを色で表現する．ただし，これでは 1 ピクセルが対応するアドレス範囲でたった 1 バイトがアクセスされた場合でも，全範囲をアクセスされた場合でも，同じように描画される．そのため，将来的にはカラースケールの調整等による使用量の表現に対応する．

ストリーミング可視化ツールではそのままアドレス軸を連続するメモリアドレス領域に対応させるが，使用ページに絞った可視化ツールでは使用ページのアドレス領域のみアドレス軸へマッピングするため各行が対応するメモリアクセス領域は不連続である．

5. 実　　装

エミュレータに valgrind を使い，ストリーミング可視化ツールと使用ページに絞った可視化ツールを実装した．

5.1 メモリアクセスログのための valgrind ツール最適化

メモリアクセスログを取得するため，lackey の `--trace-mem` オプションを改造して valgrind に新しくツールを作成した．

表 1 メモリアクセスを表現する色の対応

シアン	Store only
黄	Load only
白	Store & Load
黒	none

我々の実装ではメモリアクセスの種類を限定し，Modify を特定する機構の削除することで最適化を行った．今回の実装ではメモリアクセスの種類を Store，Load に限定して出力する．これらのメモリアクセスは，データ領域やヒープ領域等に対して行われるため，実行対象のプログラムを修正することで改善が可能であり，アプリケーション開発者にとって容易であるからである．そのためデータに対するメモリアクセスのみを取得する．また，lackey の実装ではインクリメント命令による Modify を特定するため，最後に行った 4 つのメモリアクセスを保持しておき Store がある度に確認する．処理性能改善を理由に今回の実装ではこの機構を取り除いき，メモリアクセスの種類を Store，Load とした．

さらに，メモリアクセスログの書き出しについてメモリアドレス範囲の指定機能と共有メモリへの書き出し機能を実装した．指定したメモリアドレス範囲のみを出力できることにより使用ページに絞った可視化をする際，ログ書き出しの処理性能を改善できる．共有メモリへのログ書き出しではバイナリ形式でリングバッファへ書き込むため，テキスト形式でログを書き出すよりオーバヘッドが少なくなる．ファイル出力やソケット転送の機能については既に valgrind で用意されている．

5.2 ストリーミング可視化ツール

ストリーミング可視化ツールを gtk+[8] のアプリケーションとして実装した．左に可視化領域を持ち，右上に各種情報を表示する．可視化領域では，縦軸が上から下にアドレス 0x0 から 0xffffffff に対応し，横軸は右端の列が最も新しく全体的に左へ流れながらメモリアクセスを描画する．各ピクセルでの色とメモリアクセス種類の対応を**表 1** で示す．右上に表示する各種情報には描画するアドレスの範囲と粒度がある．また，各行がどのメモリアドレスに対応しているかが分かるよう，マウスでクリックした行のメモリアドレス領域を表示する．これによりビットマップ形式による高密度な表示でも，確認したいメモリアドレスを正確に調べられる．**図 7** に概観を示す．

共有メモリを使う方法の動作としては，まず共有メモリ領域を確保し，valgrind 側からの書き込みがないか定期的に確認し，書き込みがあれば逐次描画していく．ソケット通信で

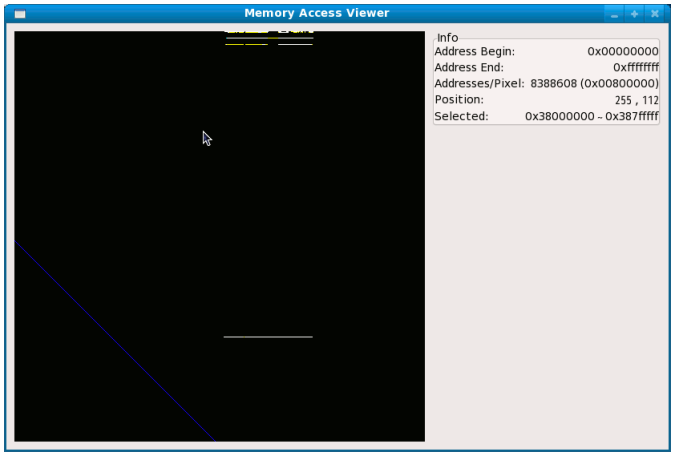

図 7 可視化ツールのモニター

はパケットを受信し，一定量取得して描画する．これらは適当な頻度で画面の更新を行うためである．

5.3 使用ページに絞った可視化ツール

ログファイルの解析，画像作成を行うプログラムを作成した．使用アドレス領域のみを可視化する．また，描画範囲や粒度を指定し任意のスケールで画像を作成できる．描画範囲はどちらの軸も指定可能で矩形の領域を指定できる．

その動作は，まずログファイルから使用アドレスのリストを作成する．現在の実装は機能を拡張できるようバイト単位にしているが，ページ単位にすることで処理性能が改善される．作成した使用アドレスリストから，指定された描画範囲と粒度でビットマップを描画する．また，出力されるビットマップの情報を保持しておく．それは 1 ピクセルあたりの命令数とメモリアドレス範囲，及びアドレス軸の各行が対応するメモリアドレスである．ユーザはこれらの情報を元にシンボルやマッピングの情報と照らし合わせ，関数の特定等が可能になる．

図 8 と**図 9** に使用ページに絞った可視化ツールにより作成した画像を示す．これは ls におけるデータへのメモリアクセスである．ページ単位で全体を俯瞰したものが図 8 であり，その一部を拡大した画像が図 9 である．それぞれの粒度は図 8 が 4096 バイト／行（ページ単位），16K 命令／列であり，図 9 は 64 バイト／行，512 命令／列である．

Vol.2010-OS-113 No.9
2010/1/28

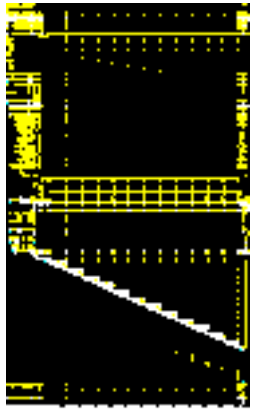

図 8 使用ページに絞った可視化ツールで
すべてのメモリアクセスをページ単位で描画.

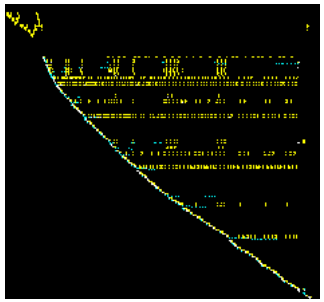

図 9 使用ページに絞った可視化ツールで
一部のメモリアクセスを拡大したもの.

6. 評　　価

エミュレータを使ったメモリアクセスログの書き出しによる性能劣化を測定した．測定を行った環境は，RAM 2GB を載せた AMD Turion（800Mhz 2 コア），OS は Fedora 10 である．

エミュレーション対象プログラムは，ls (6.12)，wc (6.12)，firefox (3.0.15) である．wc では，3MB のファイルに対してライン数をカウントする時間を測定した．firefox では，起動時間を測定した．マルチスレッドで動作する firefox の計測には，valgrind の `--trace-children` オプションを有効にすることを注意されたい．ネイティブで実行した時間に加え，エミュレーションのみを行い解析処理をしない nullgrind，及びテキスト形式でメモリアクセスを出力する lackey での実行時間を測定した．そして，我々が実装した 2 つのメモリアクセスのログ取得ツールでの処理時間を測定した．それらはデータへのメモリアクセスログをテキスト形式で出力するツール（data-text）とバイナリ形式で共有メモリへ出力するツール（data-binary-shm）である．測定結果を**表 2** に示す．

測定結果より，メモリアクセスのログをバイナリ形式で共有メモリに出力することでエミュレーションオーバヘッドに比べその処理時間が 1.07〜2.20 倍まで抑えられることがわかった．特にデータへのメモリアクセスが多い wc では lacky でのログファイル取得に比べて処理時間が 0.5%に抑えられる．このことからテキスト形式でファイル出力することが性能劣化の大きな原因であると考えられる．バイナリ形式で共有メモリへログを出力する方法では firefox のように大規模なプログラムであっても 1 分程度で起動処理が終わる程の処理性能があるため，アプリケーションの操作や画面表示を確認しながら手軽にストリーミング可視化を行える．

表 2 valgrind 出力ツールの性能測定結果（単位は秒，括弧内は nullgrind に対する処理時間の増加）

program	native	nullgrind	lackey	data-text	data-binary-shm
ls	0.007	1.18	7.26 (6.15)	3.19 (2.70)	1.26 (1.07)
wc	0.035	3.14	1330 (423)	1300 (414)	6.91 (2.20)
firefox	1.84	40.3	4910 (122)	2400 (59.6)	65.4 (1.62)

また，テキスト形式でログを出力する場合でもメモリアクセスの種類を限定することで処理性能を改善できることが確認された．ls の測定結果からメモリアクセスの種類を Load と Store に限定することで処理時間を 44%に減らせた．メモリアドレスを限定することでさらなる性能改善が期待される．

7. 今後の課題

メモリアクセスを可視化することによりその把握は直感的になった．全体を俯瞰したイメージからズームインし，バイト単位でのアクセスまで視覚的に把握できる．しかしながら，現状ではアドレスまでが分かるようになっているおり，そこがどこにマップされどのシンボルを指すのかは別途調べる必要がある．可視化ツール側でこれらの情報を取得し表示できれば分析の際に役立つ．マップやシンボル情報との連携が今後の課題として挙げられる．また，これらのメモリアクセスの分析から省メモリを実現するためにどう改善箇所を特定するか具体的に提案することが今後の課題である．本稿では Load と Store のメモリアクセスに限定したため，その他のメモリアクセスの種類についても解析を行い，ライブラリ等の使用メモリの削減についてその方法と特定手法を提案していきたい．

8. 結　　論

エミュレータによってメモリアクセスのログを取得できることに注目し，省メモリにつながるメモリアクセス解析手法としてストリーミング可視化方法及び使用ページに絞った可視化方法を提案した．インタラクティブにメモリアクセスを把握する手法とバイト単位のアク

Vol.2010-OS-113 No.9
2010/1/28

セスを調査する手法である．これらの実装と評価を行い，時系列とメモリアドレスを失わず，かつ直感的にメモリアクセスを把握できることが確認できた．また，エミュレータからのメモリアクセスログ取得について領域選択や共有メモリによるデータ渡しといった改造を行い，処理性能の向上を実現した．これによりストリーミング可視化では，アプリケーションの操作や画面表示を確認しながらメモリアクセスを把握することができるようになった．

参 考 文 献

1) : valgrind. http://valgrind.org/.
2) Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, USENIX (2005). http://www.nongnu.org/qemu/.
3) 松尾和弥，佐藤未来子，並木美太郎：QEMU を用いた命令, アドレストレーサの実現, OS 研究会報告， Vol.2007, No.36.
4) Nethercote, N. and Seward, J.: Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation, *ACM SIGPLAN Notices* (2007).
5) Yu, Y., Beyls, K. and D'Hollander, E.H.: Visualizing the Impact of the Cache on Program Execution, *In Information Visualization*, IEEE, pp.336–341 (2001).
6) Choudhury, A. I., Potter, K. C. and Parker, S. G.: Interactive Visualization for Memory Reference Traces, *Computer Graphics Forum*, Vol. 27, No. 3, pp. 815–822 (2008).
7) 安田　裕，小林良岳，中山　健，前川　守：最適なメモリ管理ポリシの自動割り当てに関する研究，SPA (2004).
8) : GTK+ 2.0 Tutorial. http://takeposo.sakura.ne.jp/unix/setting/gtk_tutorial/.