Vol.2010-SLDM-144 No.25
Vol.2010-EMB-16 No.25
Vol.2010-MBL-53 No.25
Vol.2010-UBI-25 No.25
2010/3/26

携帯端末向けプログラム再ダウンローディング方式

清原 良三[†1] 三井 聡[†2] 田中 功一[†1]
寺島 美昭[†1] 神戸 英利[†3]

携帯端末などのコンシューマ向けの機器のソフトウェア規模が巨大化し続けている．パソコンなどと違い，不具合があればソフトウェアの更新をすぐに要するなどコストがかかる．そのためなるべく不具合を少なくして出荷する必要性がある．製品の開発においてはクロス環境上のシミュレータを利用することにより効率良くデバッグ試験などを行うこともできるが，各種ネットワークを利用した場合の複合処理などタイミングに依存して起こる障害などもあり，出荷直前には実機での検証は不可欠である．出荷直前にはデバッグ，コード修正，ビルド，ダウンロード，動作確認の繰り返しになる．本論文ではこの繰り返しの中で，ダウンロード時間の短縮を目的にその高速化手法を検討し，評価をした．

Reprogramming Method for Mobile Devices

Ryozo Kiyohara,[†1] Satoshi Mii,[†2] Koichi Tanaka,[†1]
Yoshiaki Terashima[†1] and Hidetoshi Kambe [†3]

This paper presents the technology for fast downloading method during the testing and debugging phases in the development of mobile devices. The increasing number of features in mobile devices has made it difficult to release bug-free devices. Software for mobile devices has to be developed and tested using a cross-platform simulator. However, many cases have to be considered while using the target devices during testing, making several debugging phases and software update releases inevitable. These processes have to be performed iteratively. Therefore, the time required for the binary code to download to the target devices should be small. In this paper, we propose a fast downloading algorithm for these types of consumer devices.

1. はじめに

携帯端末やカーナビゲーションシステムのソフトウェアの規模が増大し，組込みソフトウェア開発の危機が叫ばれて久しい．技術者のスキルアップなど組込み技術者が養成される一方で，クロス環境上でのシミュレータの充実などデバッグの効率化なども図られている．しかしながら，携帯端末などで，複数の通信機能を搭載し，複雑な動作を状態遷移表で管理するようなケースではシミュレータで動作確認しただけでは不十分なケースも多い．そのため，出荷直前になると，多数の開発者や試験者が図1に示すように，実機に最新のソフトウェアをダウンロードしては動作試験を繰り返すということが多い[1)]．開発の最終フェーズでの効率化を目指すには，以下に示すようなアプローチがそれぞれ重要であると考える．

- 障害の発生を抑えること．
- シミュレータなどクロス環境上での開発をなるべく多くすること
- ソフトウェアのバージョンアップを高速にすること

障害の発生を抑えるためのソフトウェア開発技術は，組込み向けにも，例えば文献2),3)などの数多くの研究があり，開発手法からツール類まで製品ベースでも存在している．しかし，これらの手法で障害を完全に排除できるわけではない．

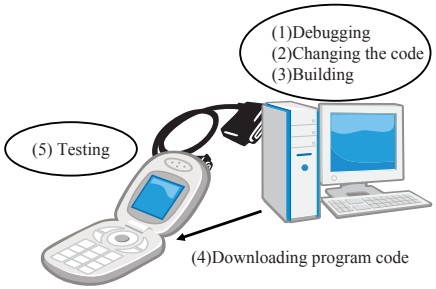

図1 デバッグ，試験，確認サイクル

†1 三菱電機（株）情報技術総合研究所
MITSUBISHI ELECTRIC CORPORATION INFORMATION TECHNOLOGY R & D CENTER
†2 三菱電機（株）名古屋製作所
MITSUBISHI ELECTRIC CORPORATION NAGOYA WORKS
†3 （株）モルフォ
Morpho, Inc.

Vol.2010-SLDM-144 No.25
Vol.2010-EMB-16 No.25
Vol.2010-MBL-53 No.25
Vol.2010-UBI-25 No.25
2010/3/26

また，シミュレータなどの環境も充実してきており，動作確認も十分可能ではあるものの，通信機能などを利用した場合のタイミング障害などの動作確認はシミュレータでは十分なわけではなく，出荷時には必ず実機で動作確認する必要があり，やはりダウンロードしての動作確認をなくすことができるわけではない．

ソフトウェアの規模が大きければダウンロードするデータ量は，フラッシュメモリへの書き込みデータ量も多くなり書き換えに時間がかかることになる．バイナリパッチで確認する手法もありえるが，動作のタイミングが本来とは変わる恐れもあり，将来のバグの元でもあるため避けるべきである．

そこで，我々はこのように何度もソフトウェアを繰り返しダウンロードする場合の高速化手法を検討し，一定の効果があることを既に示した[4)5)]．しかし，端末に付属するシリアル通信の速度や，ソフトウェアの構成方法に依存する要素も多く，既存の手法を単純に実装するだけでは，常に効果がでるわけではない．そこで，本論文では，提案アルゴリズムを様々な角度から分析し，適用するにあたり有用な情報となる指標を示す．

2. 関 連 研 究

携帯電話のソフトウェア更新技術[6)7)]を適用するとダウンロードするデータ量を削減でき，書き換え時間も短くできる．しかし，ダウンロード先のプログラムデータと同じものをサーバ上にも保持し，新しい版との差分をサーバ上で計算してから最小化した差分データを送るため，端末上のプログラムデータのバージョン管理が必須である．出荷直前で複数の人がデバッグしている環境では正しいバージョン管理をすることは困難であり，ミスを起こし易くなると想定される．そのため，このままでは適用できない．

そこで，バージョン管理をせずにしかも高速にダウンロードできる手法が必要となる．このような研究開発としては，rsync[8)9)]がある．rsyc はバージョン管理せずにネットワークを経由して複数のファイルの間での差分を転送して同期を取る技術である．ファイルを一定のブロックに分割し，ブロック単位で複数のハッシュ値を計算し，複数のハッシュ値が一致すれば同一の内容であると判断し転送しない．このようにして差分を推測し情報を転送することにより実現するが，ファイルが対象であり，そのままでは携帯電話などの組込みソフトのプログラムには適用できない．

組込みソフトのプログラム更新に rsync を適用した例として，センサーネットのノードのプログラム更新に適用する研究がある[10)11)]．これらの研究では，センサノードのマルチホップ通信を利用してプログラムの更新を行う際に rsync の適用を行う．しかし，rsync ではハッシュ計算を必要とし，センサノードのようなCPU リソースの小さい環境ではそのまま適用できないため，結局バージョン管理を行い，サーバ側で rsync のハッシュ計算を行うことにより解決している．そのため，この方式もそのまま適用はできない．

そこで，我々は既に rsync を携帯電話に組み込むための手法を提案した[4)5)]．既存の rsync との違いは変更ブロックの探し方や比較ブロックの大きさなどを環境に合わせる必要がある点などである．これらの研究では一般的に効果がありそうだということのみで，実際にどうパラメターを設定すれば効果的なのかまでは示せていなかった．

3. 提 案 手 法

3.1 アルゴリズム

我々が既に文献 4),5) に示した手法に関して説明する．図 2，図 3に示したアルゴリズム，処理シーケンスでバイナリコードを一定の分割サイズに全体を分割し，2 種類のハッシュ値を一定サイズの比較ブロックごとに計算しこれをサーバ側に送り，同じ部分を検出する．フラッシュメモリは，図 4に示すように，一旦フラッシュメモリのセクターと呼ばれる複数のページからなる消去ブロック単位に RAM に読み込んでからページ単位に書き換える．そのため，図 5のように実際にコピーを試みる際に当該データが既に書き換えられているケースを考慮しなければならない．そのため，書き換えを実施してないはずの範囲に限って行う．

2 種類のハッシュ値は端末上では計算し，これをサーバに送り，サーバ上では同一の比較ブロックを少しずつ位置をずらしながら探す．同じ部分が見つかればその部分は端末に送らない．同じ部分が見つからなかった部分のみ送る．このようにして差分を送る．なお，複数のハッシュを使ったからといった必ず同じイメージである保証はないが，文献 4),5) でも議論しているように，試験環境として利用するには十分な確率で同じデータになる．

3.2 性　　能

本方式は，表 1に示す評価環境で，表 2に示す特性を持つ A, B の種類の評価データを利用することにより，比較ブロックサイズに応じて図 6に示すような特性を持つことが既に示されている[4)5)]．パターン A では書き換え量が少ないため，比較ブロックサイズを大きくしても書き換え量は一定になる．パターン B では比較ブロックを大きくすればするほど無駄な書き換えも多くなり書き換え時間が大きくなる．しかし，これらの時間はソフトウェア構成にも依存するし，ダウンロードサイズやシリアルの転送速度とも影響しあい，実際にはどうパラメターを調整すれば最適なのかがこれだけではわからない．

Vol.2010-SLDM-144 No.25
Vol.2010-EMB-16 No.25
Vol.2010-MBL-53 No.25
Vol.2010-UBI-25 No.25
2010/3/26

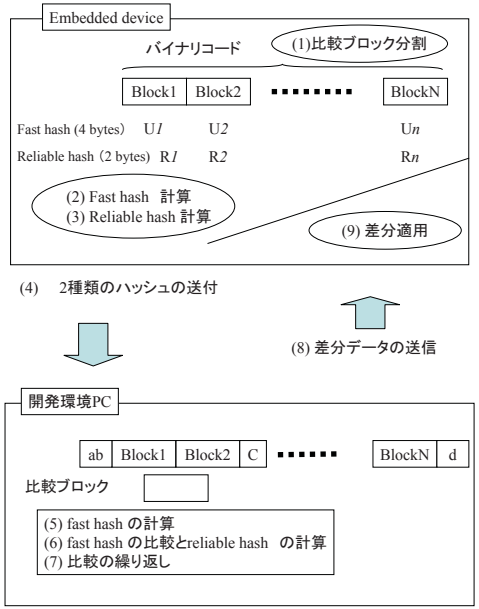

図 2 差分抽出アルゴリズム

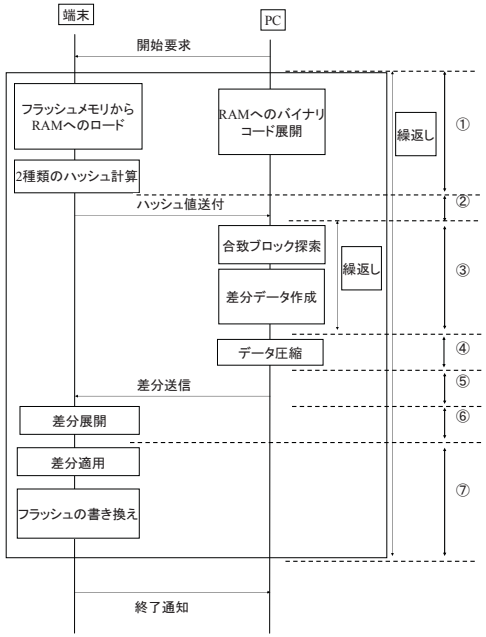

図 3 差分抽出処理シーケンス

4. 性 能 分 析

ダウンロード性能は，表3に示すような性能トレードオフがあり，シリアル転送時間や，ソフトウェア構成手法による工夫などと関係して適切なパラメターが決まる．図3に示した区間ごとの性能を検討していくため，更新パターン B を利用して詳細な時間を計測した．ここで，区間 i の所要時間を T_i とし，その区間のデバイス側所要時間を D_i，サーバ側所要時間を S_i とする．全ダウンロード時間は以下で示される．

$$DownloadTime = \sum_{i=1}^{7} T_i \tag{1}$$

ほとんどの区間の実行速度はシリアル通信や CPU 速度などの H/W の性能と比較ブロックのサイズに依存する．そのため各区間における性能を調べるため，比較ブロックのサイズを変えながら実行時間を測定した．

4.1 各区間の意味と実行時間測定結果

4.1.1 区 間 1

区間 1 は携帯端末上でのハッシュ値の計算と PC 上でのプログラムイメージのメモリへのロードが並行して動作する．端末上での計算時間の方が大きいと考えられる．従って次式で示される．

$$T_1 = max(D_1(b), S_1) \tag{2}$$

$$= D_1(b) \tag{3}$$

Vol.2010-SLDM-144 No.25
Vol.2010-EMB-16 No.25
Vol.2010-MBL-53 No.25
Vol.2010-UBI-25 No.25
2010/3/26

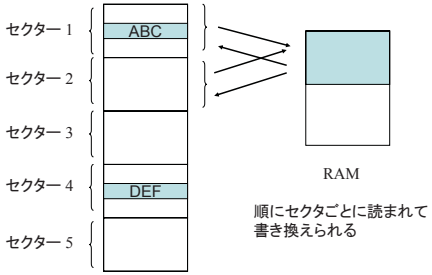

図 4 フラッシュメモリの書き換え

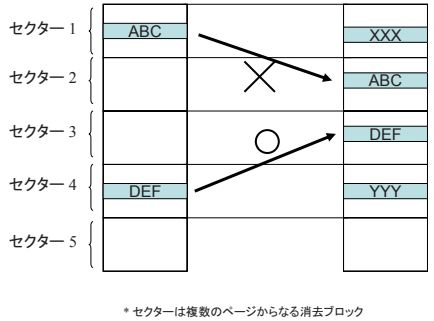

図 5 書き換え制約

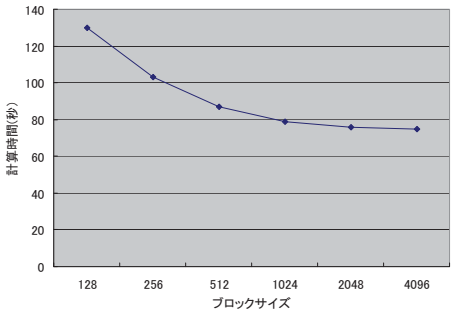

図 7 ハッシュ値計算性能

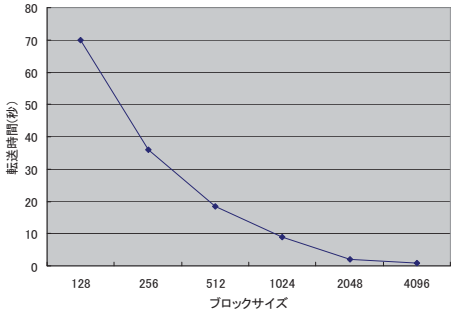

図 8 ハッシュ値転送時間

ハッシュ計算は比較ブロックの大きさに依存する要素がある．表 7にブロックサイズごとのとダウンロード時間の実測値を示した．この結果から比較ブロックが大きければ速度は速いことがわかる．

4.1.2 区 間 2

区間 2 は，携帯端末上のハッシュ値を PC に送付するため，シリアル通信速度とハッシュ値ペアの和に依存して次式で示される．ハッシュ値ペアの和のサイズは比較ブロックのサイズから決まる．図 8にブロックサイズごとの更新パタン B でのダウンロード時間比較を示す．比較ブロックが大きいほどハッシュ値転送時間が少なくて済むことがわかる．

表 1 評価環境の性能データ

項目	性能	備考
転送 PC →端末	139KB/ 秒	
転送 端末→ PC	70KB/ 秒	
圧縮 (PC)	705KB/ 秒	4Kbyte 単位圧縮
展開 (端末)	1.41MB/ 秒	4KByte 単位圧縮データの展開
フラッシュ書込	2.13MB/ 秒	
フラッシュ読込	4.27MB/ 秒	
チェックサム計算	204.08MB/ 秒	

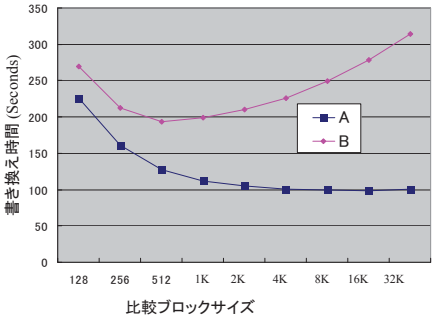

図 6 トータルダウンロード時間

表 2 評価対象データの特性

パターン名	更新消去ブロック数
A	14
B	409

表 3 性能データトレードオフ

		ブロックサイズ大	ブロックサイズ小
転送データサイズ	差分情報サイズ	大	小
	ハッシュ転送サイズ	小	大
ハッシュ衝突確率		低	高
ハッシュ計算時間		小	大

4.1.3 区 間 3

区間 3 は一致検索であるが，PC 上で動作させるため十分高速と考えて良い．実際に時間を測定しても 5 秒程度であり，区間 1, 区間 2 に比べて時間が短く，この区間を分析し高速化を図っても全体に対する影響は小さい．

4.1.4 区 間 4

区間 4 は圧縮時間であるが，貧弱なリソースしか持たない携帯端末上での解凍時間を考慮し，解凍時間が高速であると言われる byte-pair 圧縮[13)14)] を利用した．ただし， BPE は圧縮に時間がかかるため圧縮時間と展開時間のバランスを考える必要がありそうだが，実際に PC 上で計測した結果を図 9に示す．ブロックサイズは小さい方が良いのは，差分サイズが小さいからである．実行時間は PC の能力にもよるが，それほど全体からみると時間がかからないことがわかる．

4.1.5 区 間 5

区間 5 はダウンロード時間であり，差分サイズに依存する．実行時間を図 10に示す．差分サイズの小さくなる比較ブロックが小さい場合が良いことがわかる．また全体の実行時間に占める

Vol.2010-SLDM-144 No.25
Vol.2010-EMB-16 No.25
Vol.2010-MBL-53 No.25
Vol.2010-UBI-25 No.25
2010/3/26

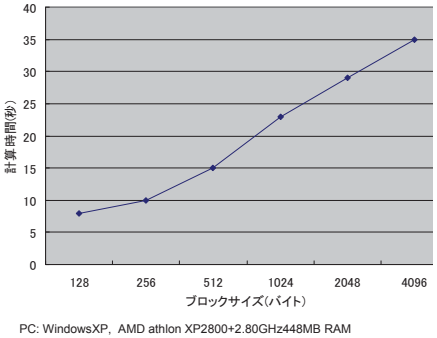

図 9 圧縮時間

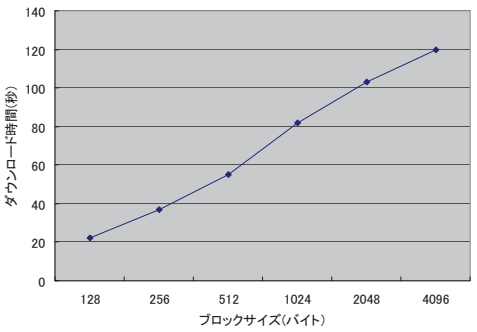

図 10 ダウンロード時間

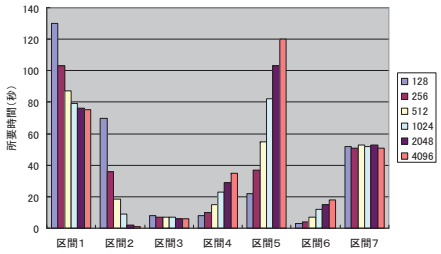

図 11 区間ごとの所要時間

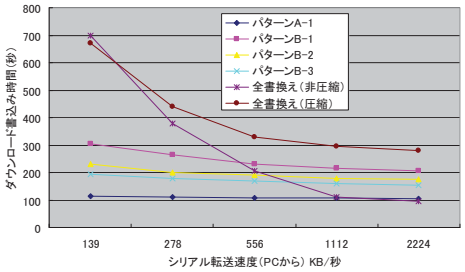

図 12 シリアル転送速度による書き込み時間の変化

割合も大きい．ここは工夫により小さくするべきである．ただし，ダウンロード時間はシリアルの速度にも依存する．

4.1.6 区　間　6

区間 6 は差分適用処理である．メモリ上のみの操作であり，時間であり，悪い場合でも 20 秒以下で全体から見ると工夫しても影響が少ない処理である．

4.1.7 区　間　7

区間 7 は書き換え処理である．本質的には書き換え処理は比較ブロックサイズには影響を受けない．比較ブロックサイズよりむしろ更新パターンとソフトウェア構成法の影響を受ける．時間も更新パターン $B50$ 秒程度はかかるため，なるべくソフトウェアの書き換えの発生する部分を小さく押さえることが重要となる．

4.2 処理全体からの分析

処理時間を比較するため，各区間ごとの所要時間を図 11にまとめた．この図から，区間 1, 区間 5, 区間 7 を重点的に時間削減できれば良いことになる．

5. 性能改善検討

5.1 区間 1 の高速化

区間 1 はハッシュ計算であり，この処理時間を短縮するためにはハッシュ値を ROM イメージ内に保持するということが考えられる．ハッシュ計算が不要になりハッシュ値を ROM に読み込むだけですみ，高速化が十分可能である．デメリットとしては毎回バイナリイメージ生成時に同時に計算をして書き込む必要があるが，PC 上での j 計算となり，実際には 5 秒もかからなかった．そのため，デメリットは ROM イメージが約 1M バイト程度大きくなってしまうことである．これが許容範囲内であれば良い．製品に依存するが，その程度の余裕はある場合が多いと考える．

ただし，通常の開発環境でのリンクの後にツールを利用してハッシュ計算をした値を実行イメージに入れる必要がある．ツールの作り方にも寄るが開発者の操作ミスや手間となっては問題である． 2 分程度の時間との比較で，開発の感じ方次第であるためどちらが良いとは言いがたい．

5.2 区間 5 の高速化

区間 5 はダウンロード時間であり，差分データとなる．ソフトウェア構成のやり方によって差分がなるべくでないようにする工夫である程度小さくできると考える．一方，シリアル通信の速度に依存すり面も大きいため，シリアル通信の速度が変わった場合の挙動をシミュレーションにより調べた結果を図 12に示す．

更新パターンの差分ファイルの他に全ファイルを圧縮してダウンロードする場合と非圧縮でダウンロードする場合とを比較のために加えた．シリアルの転送速度が速くなると圧縮せずに全ファイルを転送する方が早くなる．転送速度が速くなることにより，メモリ上での展開時間の方がより問題となるため，ある程度より早くなると圧縮せずに全情報をダウンロードするほうが良くなると考える．

一定の速度までが我々の提案方式の有効な範囲であることがわかる．

Vol.2010-SLDM-144 No.25
Vol.2010-EMB-16 No.25
Vol.2010-MBL-53 No.25
Vol.2010-UBI-25 No.25
2010/3/26

表 4 更新パターン A

パターン名	版の内容
A-1	従来の A と同じでずれはない
A-2	従来の A の修正点の直後に 16 バイトダミー追加

表 5 更新パターン B

パターン名	版の内容
B-1	従来の B にアドレスの小さいところに 16 バイトダミー追加
B-1-1	従来の B にアドレスの小さいところに 1K バイトダミー追加
B-1-2	従来の B にアドレスの小さいところに 64K バイトダミー追加
B-2	従来の B のアドレス上中央部に 16 バイトダミー追加
B-3	従来の B のアドレス上後方部に 16 バイトダミー追加

ROM イメージサイズはどの場合も約 90Mbyte

表 6 更新パターンの更新ブロック数

パターン名	更新ブロック数
A-1	14
A-2	409
B-1,B-1-1,B-1-2	699
B-2	503
B-3	409

ブロックサイズ 128K バイト

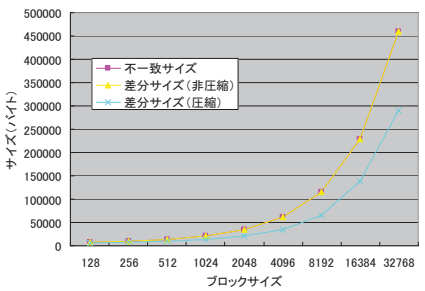

図 13 更新パターン A-1 のデータサイズ

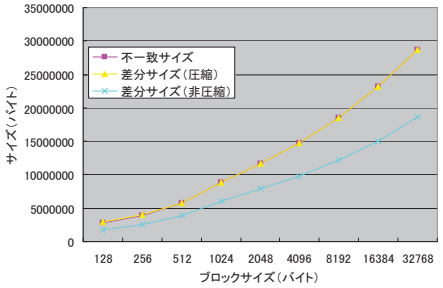

図 14 更新パターン A-2 のデータサイズ

6. ソフトウェア構成による影響

ソフトウェアの構成法によっては一部の修正でバイナリイメージ全体がずれることがあるのか，局所の修正は局所に留まることができるのかが変わる．なるべく配置が換わらないようにすることで十分効果があることがわかっている[12]．しかしながら，こういった工夫はツールを利用して出荷間際にフラッシュメモリの余裕の状態を見極めつつソフトウェアの配置を決めればできるものであり，デバッグ，試験といった繰り返しのフェーズで開発者にそこまで考えさせることはできない．しかしながら，リンカなどには ROM のアラインメント機能があり，こういった機能を活用することはちょっとした修正には有効であると考えが，一方で ROM イメージを大きくするというマイナス要素もある．そこで，前記の更新パターン A, B にさらに手を加えて，表 4,5に示す位置ずれの有無や大きさの違う更新パターンを作成し差分サイズや不一致となるデータのサイズや実行時間の評価を行った．表 6にデータの特性を示す．

6.1 位置ずれ有無による影響

位置ずれの有無による影響を調べるため，更新パターン $A-1$ と $A-2$ を利用して不一致と認識したサイズ，差分サイズの圧縮前と圧縮後および区間 3 の探索時間を測定した．測定環境としては全体を 16Mbyte 単位に分けてその間での比較を繰り返した．ブロックサイズは 128 バイトから 32K バイトを利用した．図 13,14にその結果を示す．

不一致サイズと差分（非圧縮）にはほとんど差がなく，差分を表現するためのコマンド情報などは誤差の範囲であることがわかる．比較のブロックサイズを大きくすると差分情報が大きくなりこの観点からは比較ブロックは小さい方が良い．位置ずれの有無による差は非常に大きい．なお，区間 3 における計算時間は，位置ずれのない場合で 2 秒以下． 位置ずれのある場合でも 5 秒以下であり，計算時間は気にする必要はない．

差分サイズはわずかでもずれがあると大きく影響することがわかる．しかし，前述のとおりツールなどで位置調整するのは煩わしい．しかし， 16 バイト程度であれば ROM のアラインメント機能などの活用でも十分効果があることがわかる．障害の修正がわずかな場合には ROM のアラインメント機能が有効であるが，繰り返しの修正の場合では，急激に差分サイズが増加する場合もあることになる．

6.2 位置ずれ発生箇所による影響

次に位置ずれの発生箇所の違いによる影響がどの程度か調べた．図 15に，更新パターン $B-1, B-2, B-3$ における不一致サイズをブロックサイズに応じて示した．これらの場合の差分サイズはパターン A での比率とほぼ同じ傾向を示し，区間 3 の探索時間は長くても 12 秒であり無視できる範囲であった．障害の修正位置の影響は大きく差分サイズやに影響することがわかる．また，図 16にフラッシュメモリの書き換え時間を比較ブロックサイズごとに示す．位置ずれのおきない場合は良く，位置ずれが発生場所の影響は大きく，メモリの前方の修正は避けるべきであることがわかる．

従って，試験において配置位置に制約がない場合は，開発者は自分用のリンカーでは配置をなるべく ROM のアドレス空間の大きな位置に配置することにより影響を抑えられることがわかる．

Vol.2010-SLDM-144 No.25
Vol.2010-EMB-16 No.25
Vol.2010-MBL-53 No.25
Vol.2010-UBI-25 No.25
2010/3/26

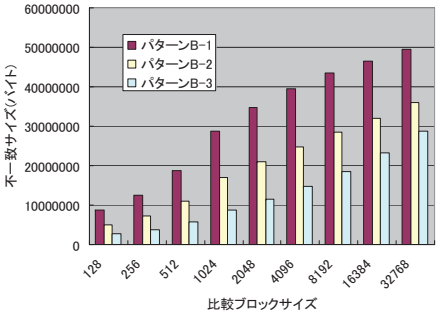

図 15 位置ずれ発生位置の影響 (差分サイズ)

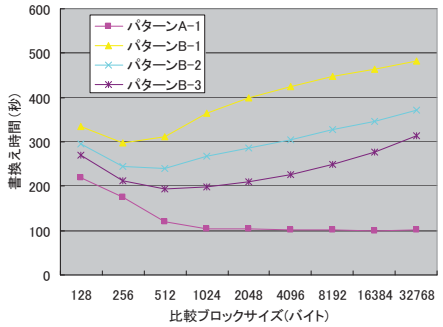

図 16 位置ずれ発生位置の影響 (書込み時間)

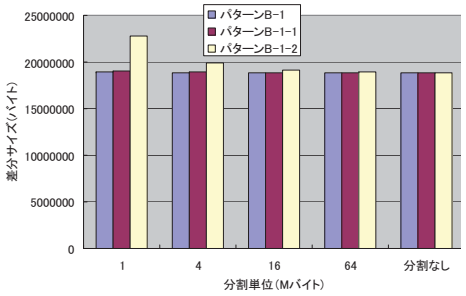

図 17 分割単位の差分サイズへの影響

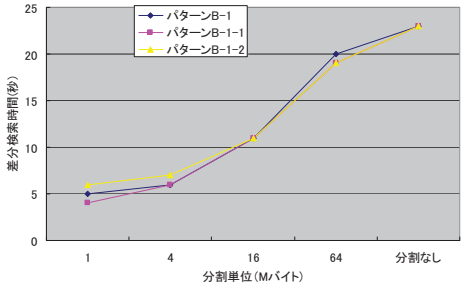

図 18 分割単位の差分検索時間への影響

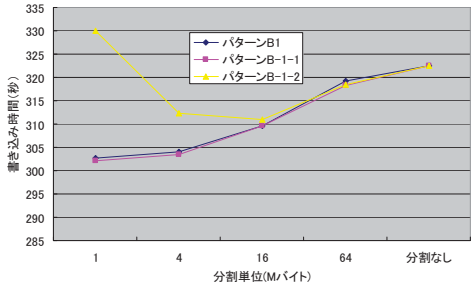

図 19 分割単位の書き込み時間への影響

6.3 分割単位による影響

rsync の適用にあたり，バイナリプログラム全体をファイルと考えるのは RAM の関係からも妥当でなく，適当な大きさのブロックに区切って比較の単位としている．この分割の単位の影響を調べた．分割する単位による影響がどの程度異なるかを示すために，図 17に差分サイズを，図 18に区間 3 である差分検索時間を示した．位置ずれ幅の大きさで比較するために更新パタン $B-1, B-1-1, B-1-2$ を利用して調べた．

結果からは位置ずれ幅が大きく，分割単位が小さい場合に差分が大きくなるものの，対象とする試験フェーズ程度の位置ずれでは分割単位の影響は差分サイズにはほとんどないと考えられる．

逆に不一致箇所の検索時間には大きく影響するため，ある程度分割単位は小さくても良いと考える．

一方，フラッシュメモリの書き換え時間を表 19に示す．ここで，位置ずれ幅の大きい場合のみ分割単位を 1M バイトとした場合に書き込み時間が大きくなっているが，これは差分サイズが大きくなっている分の影響を受けているためと考えられ，この場合に限り 16M バイトの分割が良くなるが，ここで想定するような試験サイクルでは分割単位は 16M バイト以下であれば書き込み時間と差分の検索時間を加えてもそれほど時間に差がでないのではないかと考える．

6.4 性能改善のまとめ

性能改善のためのポイントを以下にまとめる．

(1) 位置ずれが発生しないようにソフトウェアを構成すること．しかし，位置ずれの発生を予測することは無理である．そのため，位置ずれが発生したとしても影響範囲が小さくなるように自分の担当モジュールなどは ROM 上の後方に配置しておくこと．また，リンカなどによるアラインメント機能は活用し，最低限の位置固定の努力はしておくことが望ましい．

(2) 比較ブロックサイズを適切に決めること．比較ブロックサイズを開発者がダイナミックに決めることは難しい．従って予め適切な値に固定することが望ましいが，場合によってその適切な値は変わる．そこで，多くの場合は早いが，最悪ケースは遅くなっても良いのであれば，なるべく比較ブロックは大きくし， 16K バイト程度が良い．しかし，最悪ケー

Vol.2010-SLDM-144 No.25
Vol.2010-EMB-16 No.25
Vol.2010-MBL-53 No.25
Vol.2010-UBI-25 No.25
2010/3/26

スでもそれなり速度を期待するのであれば，256 バイト程度の小さな値にすると良い．バランス的なものを考慮すると 512 バイトから 1K バイト程度が妥当であると考える．

7. おわりに

携帯端末の開発フェーズ終盤での実機 H/W へのソフトウエアダウンロードの高速化手法に関して提案，評価した．提案手法は，モバイル端末などにおけるファイル同期化手法の中で有効な手法である rsync のアルゴリズムをベースにした手法を適用することで効果があることを示した．rsync を単純に適用するのではなく，プログラム位置の固定と適当なサイズに比較ブロックサイズを選ぶことが重要であることを示した．

PC 側にハッシュ計算などのツールを使うことがあるのであれば，修正内容を調査し，影響範囲まで調べるツールの導入も考えられる．即ちバージョンは管理しないが想定したバージョンが端末に入っていれば高速に書き換えられるが，そうでなくとも正しく書き換えられるという方法の導入が考えられる．

今後，このような手法の導入と実際にそれで短くなる時間との効果を検討しながら開発を進めるべきと考える．

参考文献

1) Cusumano M., MacCormack A., Kemerer F. Chris and et al.: Software Development Worldwide: The State of the Practice, IEEE Software, Vol.20,No.6,pp.28-34(2003)
2) Mellor J. S. and Balcer M., "Executable UML: A Foundation for Model- Driven Architecture," Addison-Wesley, 2002.
3) Mellor J. S. and Clark N. A., and Futagami T., "Model-Driven Development," IEEE Software, Vol.20, No.5, pp.14-18, 2003.
4) 清原良三，三井聡，神戸英利ほか，"端末開発における開発効率化の一検討,"情報処理学会研究報告,Vol.2008,No.107,2008-MBL-047,pp.1-8(2008)
5) Kiyohara R., Mii S., Tanaka K., et al., "Study on Binary Code Synchronization in Consumer Devices," IEEE on Transs.CE, Vol.56, Issue 1, (2010)(to appear)
6) Kiyohara R., Kurihara M., Mii S., et al., "A Delta Representation Scheme for Updating between Versions of Mobile Phone Software," Electronics and Communications in Japan, Vol.90, No.7, pp.26-37, 2007.
7) Terazono K. and Okada Y.: An Extended Delta Compression Algorithm and the Recovery of Failed Updating in Embedded Systems, Proc. IEEE Data Compression Conference 2004,p.571(2004)
8) Tridgell A. and MacKerras P., "The rsync algorithm," Technical Report TR-CS-96-05, Australian National University, 1996.
9) Rsync: http://rsync.samba.org
10) Pamta K. R., Bagchi S., and Midkiff P. S., "Zephyr: Efficient Incremental Reprogramming of Sensor Nodes using Function Call Indirections and Difference Computation," Usenix 2009
11) Pamta K. R. and Bagchi S., "Hermes: Fast and Energy Efficient Incremental Code Updates for Wireless Sensor Networks, " Proc.IEEE INFOCOM 2009, pp.639-647, 2009
12) 清原良三，栗原まり子，古宮章裕ほか: 携帯電話ソフトウェアの更新方式，情報処理学会論文誌，vol.46，no.6，pp.1492-1500(2005).
13) Gage P., "A New Algorithm for Data Compression," The C Users Journal, Vol.12, No.2, (1994), 23.38.
14) Shibata Y., Matsumoto T., Takeda M., et al., "A Boyer-Moore type algorithm for compressed pattern matching," Proc. 11th annual Symposium on Combinatorial Pattern Matching, Vol.1848 of Lecture Notes in Computer Science, (2000), 181.194.